公告

在互联网某公司混迹6年，也算是互联网的一个老兵。从事java开发有8年有余，也算是老古董了一个。对web开发有浓厚的兴趣。
微博:GodIsCoder
独立Blog: God Is Coder
个人网站: iphone发码网
昵称: God Is Coder
园龄：4年6个月
荣誉：推荐博客
粉丝：791
关注：11
+加关注

java中volatile关键字的含义

在java线程并发处理中，有一个关键字volatile的使用目前存在很大的混淆，以为使用这个关键字，在进行多线程并发处理的时候就可以万事大吉。

Java语言是支持多线程的，为了解决线程并发的问题，在语言内部引入了 同步块 和 volatile 关键字机制。

synchronized

同步块大家都比较熟悉，通过 synchronized 关键字来实现，所有加上synchronized 和 块语句，在多线程访问的时候，同一时刻只能有一个线程能够用 synchronized 修饰的方法 或者 代码块。

volatile

用volatile修饰的变量，线程在每次使用变量的时候，都会读取变量修改后的最的值。volatile很容易被误用，用来进行原子性操作。

下面看一个例子，我们实现一个计数器，每次线程启动的时候，会调用计数器inc方法，对计数器进行加一

执行环境——jdk版本：jdk1.6.0_31 ，内存 ：3G cpu：x86 2.4G

```
1  public class Counter {
2
3      public static int count = 0;
4
5      public static void inc() {
6
7          //这里延迟1毫秒，使得结果明显
8          try {
9              Thread.sleep(1);
10         } catch (InterruptedException e) {
11         }
12
13         count++;
14     }
15
16     public static void main(String[] args) {
```

最新评论

1. Re:java中volatile关键字的含义

代码修改如下，结果是始终等于N次了。修改包括：1）保证了所有的线程结束后再打印结果；2）共享变量修改为AtomicInteger，从而保证自增操作的原子性；虽然现在达到目的了，但还有一个疑问，我现在没.....

--阿磊2016

2. Re:java中volatile关键字的含义

@JaredYang引用我写个例子吧：楼主的列子有误！public class VolatileDemo { private static volatile int count; priv.....

--阿磊2016

3. Re:java中volatile关键字的含义

这篇本来讲的很简单的一件事，感觉评论都把人弄晕了。楼主想表达的就是volatile关键字不能保证i++操作的原子性，69楼举的例子就解释了，上面帖子最后一张线程工作内存的图就说明了不加volatile.....

--windranger

4. Re:java中volatile关键字的含义

我认为楼主并没有写错，你们那些有问题的直接在那个方法里面把那个count输出

```

        //同时启动1000个线程，去进行i++计算，看看实际结果

        for (int i = 0; i < 1000; i++) {
            new Thread(new Runnable() {
                @Override
                public void run() {
                    Counter.inc();
                }
            }).start();
        }

        //这里每次运行的值都有可能不同,可能为1000
        System.out.println("运行结果:Counter.count=" + Counter.count);
    }
}
```

```

```

```
运行结果:Counter.count=995
```

```
实际运算结果每次可能都不一样，本机的结果为：运行结果:Counter.count=995，可以看出，在多线程的环境下，Counter.count并没有期望结果是1000
```

```

```

```
很多人以为，这个是多线程并发问题，只需要在变量count之前加上volatile就可以避免这个问题，那我们在修改代码看看，看看结果是不是符合我们的期望
```

```
public class Counter {

    public volatile static int count = 0;

    public static void inc() {

        //这里延迟1毫秒，使得结果明显
        try {
            Thread.sleep(1);
        } catch (InterruptedException e) {
        }

        count++;
    }

    public static void main(String[] args) {

        //同时启动1000个线程，去进行i++计算，看看实际结果

        for (int i = 0; i < 1000; i++) {
            new Thread(new Runnable() {
```

来不就可以了？ 你看看，他有没有每次加到1000的？答案是没有！根本不需要再main里面去输出，有些人还以为是main里面的......

--不是咸鱼

5. Re:java嵌套类(Nested Classes)总结

“对于local inner Class，局部类可以表示为A$1B的方式，其中A为外部类，B为第一个局部类 如果在不同的方法里面定义了同名的局部类B，编译器是可以编译通过的，那么定义的第二个局部类B可......

--迷路的猫

6. Re:java嵌套类(Nested Classes)总结

不错，有些东西还是第一次接触到，学习中......

--迷路的猫

7. Re:java中volatile关键字的含义

看了博文+评论，学习了

--ZH奶酪

8. Re:java中volatile关键字的含义

1.方法 加同步 public static synchronized void inc() {
2.结果获取前 延迟 2s

结果就会是1000.。。

--nalia

9. Re:java中volatile关键字的含义

你怎么不延迟1秒呢，明显你这是用已经要结束的进程去求还未运行结束的线程的结果。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

--思忆未央

10. Re:java中volatile关键字的含义

在打印之前睡两秒！
Thread.sleep（2000）；
System.out.println("运行结果:Counter.count=" +

```java
                @Override
                public void run() {
                    Counter.inc();
                }
            }).start();
        }

        //这里每次运行的值都有可能不同,可能为1000
        System.out.println("运行结果:Counter.count=" + Counter.count);
    }
}
```

运行结果:Counter.count=992

运行结果还是没有我们期望的1000，下面我们分析一下原因

在 java 垃圾回收整理一文中，描述了jvm运行时刻内存的分配。其中有一个内存区域是jvm虚拟机栈，每一个线程运行时都有一个线程栈，

线程栈保存了线程运行时候变量值信息。当线程访问某一个对象时候值的时候，首先通过对象的引用找到对应在堆内存的变量的值，然后把堆内存

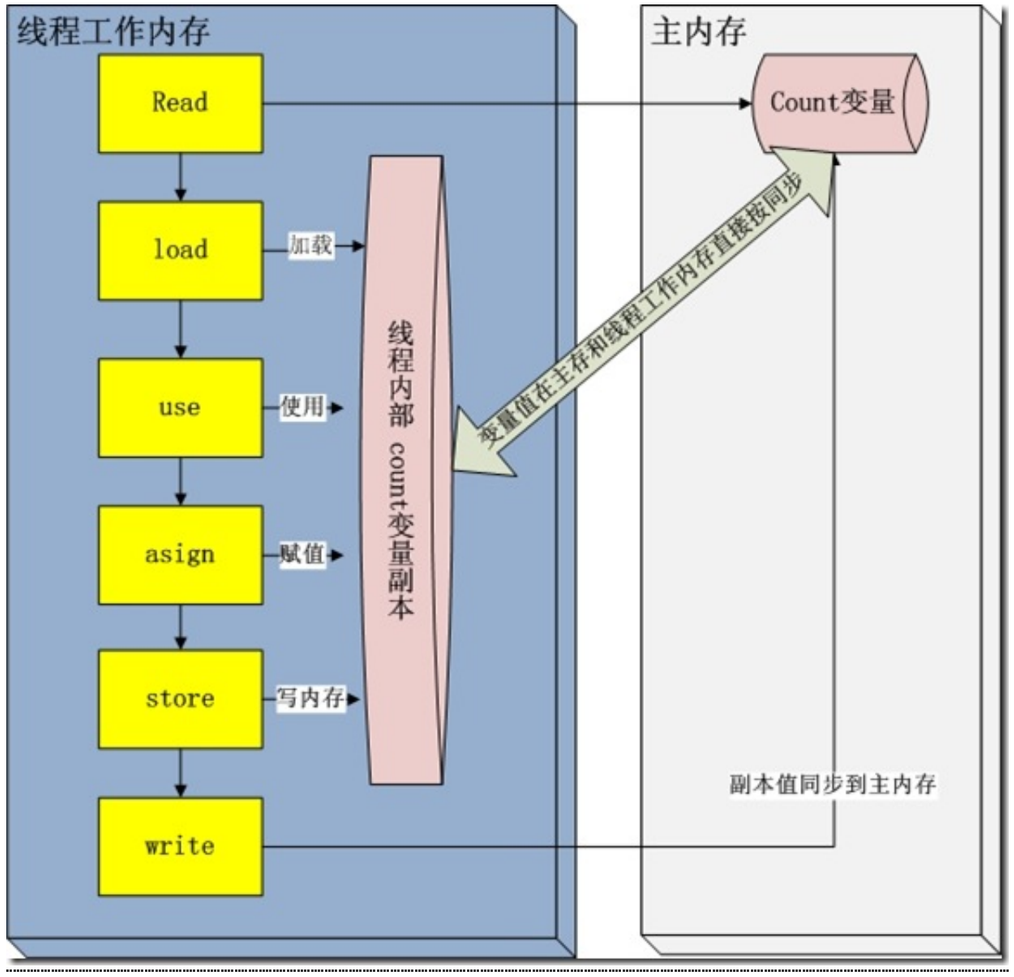
变量的具体值load到线程本地内存中，建立一个变量副本，之后线程就不再和对象在堆内存变量值有任何关系，而是直接修改副本变量的值，

在修改完之后的某一个时刻（线程退出之前），自动把线程变量副本的值回写到对象在堆中变量。这样在堆中的对象的值就产生变化了。下面一幅图

描述这写交互

Counter.count);直接打印，当然不是1000了.....

--小子千金



read and load 从主存复制变量到当前工作内存
use and assign 执行代码，改变共享变量值
store and write 用工作内存数据刷新主存相关内容

其中use and assign 可以多次出现

但是这一些操作并不是原子性，也就是 在read load之后，如果主内存count变量发生修改之后，线程工作内存中的值由于已经加载，不会产生对应的变化，所以计算出来的结果会和预期不一样

对于volatile修饰的变量，jvm虚拟机只是保证从主内存加载到线程工作内存的值是最新的

例如假如线程1，线程2 在进行read,load 操作中，发现主内存中count的值都是5，那么都会加载这个最新的值

在线程1堆count进行修改之后，会write到主内存中，主内存中的count变量就会变为6

线程2由于已经进行read,load操作，在进行运算之后，也会更新主内存count的变量值为6

导致两个线程及时用volatile关键字修改之后，还是会存在并发的情况。

分类: java语言

标签: java volatile

« 上一篇：培训测试驱动开发有感
» 下一篇：java 垃圾回收总结（1）

posted @ 2012-04-01 17:56 God Is Coder 阅读(211221) 评论(89) 编辑 收藏

< Prev 1 2

评论列表

#51楼 2015-09-22 10:49 wanwusheng

```
package com.lianglin.test;
public class VolatileTest {
static class MyObject{
static int mycount=0;
}
public static void inc() {
MyObject.mycount++;
}
public static void main(String[] args) {

//同时启动1000个线程，去进行i++计算，看看实际结果

for (int i = 0; i < 1000; i++) {
new Thread(new Runnable() {
@Override
public void run() {
inc();
}
}).start();
}
```

```
try {
Thread.sleep(2000);
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
};

//这里每次运行的值都有可能不同,可能为1000
System.out.println("运行结果:Counter.count=" +MyObject.mycount);
}
}
```

每次输出值都是：1000

对于值引用来说，多线程操作的是变量的副本，操作完后刷新到主存中。而对于地址引用，多线程是通过地址操作的是同一个变量。volatitle关键字告诉编译器，直接去通过地址操作变量，而不是变量的副本

支持(6) 反对(5)

#52楼 2015-09-29 11:32 edeef

@wanwusheng
如果将 inc方法改为:

```
public static void inc() {

for(int i = 0; i < 100; i++){
count++;
}
}
```

同样是不能保证每次都是10*1000的值，
你前面那个能每次保证1000，估计是因为for循环里面的线程是顺序创建调用的和每次执行一次，才出现每次那样的结果

支持(0) 反对(0)

#53楼 2015-10-19 20:47 docNET菜鸟

Java中的原子操作包括：
1）除long和double之外的基本类型的赋值操作
2）所有引用reference的赋值操作
3）java.concurrent.Atomic.* 包中所有类的一切操作
count++不是原子操作，是3个原子操作组合
1.读取主存中的count值，赋值给一个局部成员变量tmp
2.tmp+1
3.将tmp赋值给count
可能会出现线程1运行到第2步的时候，tmp值为1；这时CPU调度切换到线程2执行完毕，count值为1；切换到线程1，继续执行第3步，count被赋值为1----------结果就是两个线程执行完毕，count的值只加了1；
还有一点要注意，如果使用AtomicInteger.set(AtomicInteger.get() + 1)，会和上述情况一样有并发问题，要使用AtomicInteger.getAndIncrement()才可以避免并发问题

支持(2) 反对(0)

#54楼 2015-11-24 13:51 rachel0614

听同事说csdn上少有靠谱的帖子。

支持(0) 反对(0)

#55楼 2015-11-30 17:24 SUNAJING

@ wanwusheng
第一次执行的时候是999.执行多次才是1000

支持(0) 反对(0)

#56楼 2015-12-12 17:05 老金

楼主理解严重有问题，你这个程序出问题的原因是自增操作不是原子操作所以会混乱，当然了 假设自增操作是系统原子操作（不是显示加锁） 也可能会不对 这时候就是volatile的问题了 因为非volatile变量有可能会被缓存导致其他线程读取脏数据。 楼主还是好好看看多线程

支持(0) 反对(4)

#57楼 2015-12-22 16:13 总有刁民想害Zhen

```
public static void main(String[] args) {
    Thread threads[] = new Thread[1000];
    int times = 1000;
    for (int i = 0; i < times; i++) {
        threads[i] = new Thread(new Runnable() {

            @Override
            public void run() {
                VolatileTest2.inc();
            }
        });
        try {
            threads[i].start();
            threads[i].join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    System.out.println("VolatileTest2.count final:" + VolatileTest2.count);
}
```

每次start后同时要join一下，这么做最后就应该会输出想要的结果了把，但是这样就没有了并发的操作，相当于队列执行，这就没有同时操作一个变量的概念了，只是打个酱油....

支持(1) 反对(1)

#58楼 2015-12-28 14:06 1+1==2

```
package com.bizfocus.cn;

public class MultThread{
    private static int count=0;
    private static int count2=0;
    public static void inc(){
        for(int i=0;i<=10000;i++){
            count++;
        //  System.out.println(count);//注释掉这句和加上这句区别
        }
    }
    public static void main(String[] args) throws InterruptedException{

        new Thread(
```

```
            new Runnable() {

                public void run() {
                    inc();
                }
            }
        ).start();
        for(int i=0;i<=10000;i++){
            count2++;
        }

        System.out.println(count+"   "+count2);
    }
}
```

出现这个问题原因之一是因为main线程比其中的某些线程先执行完，因此显示的并不是全部线程执行完后的答案。

支持(4) 反对(0)

#59楼 2015-12-29 10:43 wangnwpu

楼主的例子的确有不恰当的地方，不过“volatile 变量不能像 synchronized 那样普遍适用于实现线程安全”的观点是对的。

支持(0) 反对(0)

#60楼 2015-12-29 14:55 BeanMr

感觉楼主描述与我的理解有所冲突，volatile绝对保证了代码获取的是内存中最新的值，JVM保证了在volatile值写入之前的操作对其它线程课件的happen-before。这一点可以通过查看每个线程拿到的原始值都是不同的来印证，出现问题的是因为没有同步机制保证最后是哪个线程写入的被main读取的值。

支持(0) 反对(0)

#61楼 2015-12-29 14:58 BeanMr

@
我的意思是 不确定main读到的是哪个线程写入的。如果main线程一直等待所有线程完成后那么值总是1000

支持(0) 反对(1)

#62楼 2015-12-29 15:03 BeanMr

```
public class VolatileCache {
    private static final int threadQuantity = 1000;
    public static CountDownLatch startLatch = new CountDownLatch(threadQuantity);
    public static CountDownLatch endLatch = new CountDownLatch(threadQuantity);
    public volatile static int count = 0;

    public static void inc() {
        count++;
    }


    public static void main(String[] args) {
```

```
        //同时启动1000个线程，去进行i++计算，看看实际结果
        for (int i = 0; i < threadQuantity; i++) {
            new Thread(new Runnable() {
                public void run() {
                    String name = Thread.currentThread().getName();
                    try {
                        startLatch.await();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    int got = VolatileCache.count;
                    VolatileCache.count = got + 1;
                    System.out.println(got + ",->," + VolatileCache.count + "," + name);
                    endLatch.countDown();
                }
            }, "Thread-" + i).start();
            startLatch.countDown();
        }
        try {
            System.out.println("Waiting Work Done");
            endLatch.await();
            System.out.println("运行结果:Counter.count=" + VolatileCache.count);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

支持(1) 反对(0)

#63楼 2016-01-27 12:59 warboy

楼主真是在一本正经的胡说八道啊

支持(2) 反对(1)

#64楼 2016-01-28 16:27 wuerbuzuo

@ Java菜鸟努力高飞
我把main的线程休眠3秒也是一样的效果

支持(0) 反对(0)

#65楼 2016-01-28 16:34 wuerbuzuo

@ ocaicai
试过了 效果一样不是1000

支持(0) 反对(0)

#66楼 2016-02-06 10:57 小白哥哥啊

服了。。这也行。。

支持(0) 反对(0)

#67楼 2016-02-19 17:36 小小陌上花开

@ 蓝色D幻想
改成原子操作，也要等子线程执行完，不然数据也不对的

支持(0) 反对(0)

#68楼 2016-02-26 11:37 -S-

结论是啥？

支持(0) 反对(0)

#69楼 2016-02-26 11:47 -S-

对volatile声明的变量单个调用时保持原子性，复合调用时代码不具备原子性，网上有一段代码：

```
class VolatileFeaturesExample {
    volatile long vl = 0L;  //使用volatile声明64位的long型变量

    public void set(long l) {
        vl = l;   //单个volatile变量的写
    }

    public void getAndIncrement () {
        vl++;    //复合（多个）volatile变量的读/写
    }


    public long get() {
        return vl;   //单个volatile变量的读
    }
}
```

上述代码在多线程调用时类似下面：

```
class VolatileFeaturesExample {
    long vl = 0L;               // 64位的long型普通变量

    public synchronized void set(long l) {     //对单个的普通 变量的写用同一个监视器同步
        vl = l;
    }

    public void getAndIncrement () { //普通方法调用
        long temp = get();           //调用已同步的读方法
        temp += 1L;                  //普通写操作， <b>这里不具备原子性</b>
        set(temp);                   //调用已同步的写方法
    }
    public synchronized long get() {
    //对单个的普通变量的读用同一个监视器同步
```

```
        return vl;
    }
}
```

支持(1) 反对(0)

#70楼 2016-03-01 18:13 斌斌有理

@ wanwusheng
static class MyObject {
static int mycount = 0;
}

public static void inc() {
try {
Thread.sleep(1);
} catch (InterruptedException e) {
}
MyObject.mycount++;
}
你加个延迟试试

支持(0) 反对(0)

#71楼 2016-03-24 17:55 密斯特雷

@ wanwusheng
大哥，你别误人子弟！你试试循环到10000，看看结果是多少！！

支持(0) 反对(0)

#72楼 2016-04-06 17:43 nowto

博主写一个可以同步起来，无论怎么运行结果都是1000的，出来呗

支持(0) 反对(0)

#73楼 2016-04-11 16:15 Paradely

@ wanwusheng
你这个程序保证了每个线程的运行是有序的，相当于顺序执行，不能说明问题。

支持(0) 反对(0)

#74楼 2016-04-13 11:18 wanwusheng

@Paradely,@密斯特雷,程序确实不能说明问题。经验证volitile是不能保证原子性，代码如下：

```
public class Test {
    static  class MyObject {
        static  int mycount = 0;
        public static  void inc(CountDownLatch latch) {
            MyObject.mycount++;
            latch.countDown();
        }
    }

    public static void main(String[] args) throws Exception{

```

```
12          final CountDownLatch latch=new CountDownLatch(10000);
13          for (int i = 0; i < 10000; i++) {
14              new Thread(new Runnable() {
15
16                  @Override
17                  public void run() {
18                      MyObject.inc(latch);
19                  }
20              }).start();
21          }
22
23          latch.await();//等待所有工人完成工作
24          System.out.println("运行结果:Counter.count=" + MyObject.mycount);
25      }
26  }
```

支持(0) 反对(0)

#75楼 2016-04-14 17:23 一只哈士奇

跑了几次，结果9996 9992

支持(0) 反对(0)

#76楼 2016-04-21 13:51 ddsdggsg

真是长期更新的评论啊.....

支持(0) 反对(0)

#77楼 2016-05-23 09:30 hcjzsw

确实是长期更新的评论

支持(0) 反对(0)

#78楼 2016-05-28 09:14 猿类的进化史

此文误导人，楼主还不明白并发的概念

支持(0) 反对(0)

#79楼 2016-05-30 16:31 RedPersimmon

@ wanwusheng
本地运行，1000的时候多，但是还是会有其他的值出现

支持(0) 反对(0)

#80楼 2016-05-30 16:48 RedPersimmon

@ 总有刁民想害Zhen
赞，加入join试了下是可以的，目前将并发线程数目提高到100000，发现加入join和不加join区别还是有的

支持(0) 反对(0)

#81楼 2016-06-13 16:30 AndKid

www.cnblogs.com/dolphin0520/p/3920373.html

这篇文章道出了真相，楼主说的没问题
volatile保证了可见性，即一个线程修改了inc，那么其他线程在下次读取inc的时候会从内存读取而不是高速缓存。但对于已经读取inc的线程就无能为了了，即没有保证整个读取、修改、回写的原子性。

支持(0) 反对(0)

#82楼 2016-06-29 11:25 小子千金

在打印之前睡两秒！
Thread.sleep（2000）；
System.out.println("运行结果:Counter.count=" + Counter.count);

直接打印，当然不是1000了！有时间差好不好！

支持(1) 反对(0)

#83楼 2016-07-19 17:19 思忆未央

你怎么不延迟1秒呢，明显你这是用已经要结束的进程去求还未运行结束的线程的结果。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

支持(0) 反对(0)

#84楼 2016-08-03 14:40 nalia

1.方法 加同步 public static synchronized void inc() {
2.结果获取前 延迟 2s

结果就会是1000.。。

支持(0) 反对(0)

#85楼 2016-08-22 17:54 ZH奶酪

看了博文+评论，学习了

支持(0) 反对(0)

#86楼 2016-09-16 21:04 不是咸鱼

我认为楼主并没有写错，你们那些有问题的直接在那个方法里面把那个count输出来不就可以了？ 你看看，他有没有每次加到1000的？答案是没有！
根本不需要再main里面去输出，有些人还以为是main里面的线程没有等到子线程修改完才读取！楼主应该没问题，就是这个demo写的容易造成大家曲解！
代码如下：

```
public static void inc() {

//这里延迟1毫秒，使得结果明显

count++;

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

支持(0) 反对(1)

#87楼 2016-09-27 14:07 windranger

这篇本来讲的很简单的一件事，感觉评论都把人弄晕了。楼主想表达的就是volatile关键字不能保证i++操作的原子性，69楼举的例子就解释了，上面帖子最后一张线程工作内存的图就说明了不加volatile关键字的情况，如果加了volatile JMM就会强制线程读\写操作直接跟主内存交互，禁用线程工作内存，但是i++这种操作还会有个temp变量作为中间过渡，以致于导致线程不安全。推荐看《深入理解java内存模型》系列文章。

支持(1) 反对(0)

#88楼 2016-10-03 15:24 阿磊2016

@ JaredYang
引用

我写个例子吧：楼主的列子有误！
public class VolatileDemo {

private static volatile int count;
private static CountDownLatch countDownLatch = new CountDownLatch(100);

public static void main(String[] args) throws Exception{
for (int i=0;i<100;i++){
new Thread(){
...

当数字改大后出现问题的概率会增加，例如改为1000试几次。

支持(0) 反对(0)

#89楼 2016-10-03 16:00 阿磊2016

代码修改如下，结果是始终等于N次了。修改包括：
1）保证了所有的线程结束后再打印结果；
2）共享变量修改为AtomicInteger，从而保证自增操作的原子性；

虽然现在达到目的了，但还有一个疑问，我现在没有用把共享变量声明为volatile，为什么没有主内存和线程工作内存不同步的问题，例如多个线程同时读到了数据，然后各自在工作线程内修改（尽管是原子的），然后各自写回到主内存，不应该有覆盖的情况吗？难道Atomic变量天生就是volatile的？

```
public class Counter4 {

    public static AtomicInteger count = new AtomicInteger(0);

    public static void inc() {

        //这里延迟1毫秒，使得结果明显
        try {
            Thread.sleep(1);
        } catch (InterruptedException e) {
        }

        count.getAndIncrement();
    }

    public static void main(String[] args) throws InterruptedException {

        //同时启动1000个线程，去进行i++计算，看看实际结果
        int N = 100000;
        ExecutorService pool = Executors.newFixedThreadPool(20);
        for (int i = 0; i < N; i++) {
            Runnable runnable = new Runnable(){
                @Override
                public void run() {
                    Counter4.inc();
                }
```

```
            };
            pool.submit(runnable);
        }

        pool.shutdown();
        pool.awaitTermination(50, TimeUnit.SECONDS);

        //这里每次运行的值都有可能不同,可能为1000
        System.out.println("运行结果:Counter.count=" + Counter4.count);
    }
}
```

支持(0) 反对(0)

< Prev 1 2

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

最新知识库文章：

· 陈皓：什么是工程师文化？
· 没那么难，谈CSS的设计模式
· 程序猿媳妇儿注意事项
· 可是姑娘，你为什么要编程呢？
· 知其所以然（以算法学习为例）

» 更多知识库文章...