ava执行CAS的实践者Unsafe类，该类中的方法都 用为主介绍Unsafe类，最后再介 包中的Atomic系统使用CAS原理实现的

而言，他们认为事情总会往好的方向发展，总是认 顾忌地做事，但对于悲观派而已， 会认为发展事态如果不及时控制，以后
映射到并发编程中就如同加锁与无锁的策略，即加 因为对于加锁的并发程序来说， 是认为每次访问共享资源时总会发生冲
对共享资源的访问没有冲突，线程可以不停执行， 略则采用一种称为CAS的技术 线程执行的安全性，这项CAS技术就是

下

明已经有其他线程做了更新，则当前线程什么都不 共一个期望值，当期望值与当前线程的变量值相同时，说明还没线程修改该值，前线程不符，则说明该值已被其他线程修改，此时 变量再尝试再次修改该变量，也可以放弃操作，原理图如下

当多个线程同时使用CAS操作一个变量时，只有一个会胜出，并成功更新，其余均会失败，但失败的线程并不会被挂起，仅是被告知失败，并且允许再次尝试，当然也允许，CAS操作即使没有锁，同样知道其他线程对共享资源操作影响，并执行相应的处理措施。同时从这点也可以看出，由于无锁操作中没有锁的存在，因此不可能出现死锁的

并且CAS的步骤很多，有没有可能在判断V和E相同后，正要赋值时，切换了线程，更改了值。造成了数据不一致呢？答案是否定的，因为CAS是一种系统原语，原语属于一个过程，并且原语的**执行必须是连续的**，在执行过程中不允许被中断，也就是说CAS是一条CPU的原子指令，不会造成所谓的数据不一致问题。

计一样直接操作内存，单从名称看来就可以知道该 的指针操作，因此总是不应该首先使用Unsafe类，Java官方也不建议直接使用
的执行依赖于Unsafe类的方法，注意Unsafe类中 nsafe类中的方法都直接调用操作系统底层资源执行相应任务，关于Unsafe类的

;

s, byte value);//设置给定内存地址的值public native void putAddress(long address, long x);

9

8

te及getByte相同

```
ntiationException;
```

该变量的内存地址，

9

8

trap类加载器使用，普通用户调用将抛出异常，所以我 象并进行相关操作。

中无锁操作CAS基于以下3个方法实现，在稍后讲解Atomic系列内部方法是基于下述方法的实现的。

通过这个偏移量迅速定位字段并设置或获取该字段的值，
邪通过CAS原子指令执行操作。

```
(Object o, long offset,Object expected, Object x);

ject o, long offset,int expected,int x);

oject o, long offset,long expected,long x);
```

直到超时或者中断等条件出现。unpark可以终止一个挂起的线程，使其恢复正常。Java对线程的挂起操作被封装在 LockSupport类中，LockSupport类中有各种版本pack方法，其底层实现最终

条件出现。

```
 time);
```

挂起操作都是在LockSupport类实现的，其底层正是使用这两个方法，

9

8

针类Unsafe类有了比较全面的认识，下面进一步分 子操作类(Atomic系列)，从JDK 1 提供了`java.util.concurrent.atomi`
性能高效，主要分以下4种类型。

列进行分析，AtomicInteger主要是针对int类型的数据执行原子操作，它提供了原子自增方法、原子自减方法以及原子赋值方法等，鉴于AtomicInteger的源码不多，我们直接看源码

```
nts java.io.Serializable {
 6214790243416807050L;


getUnsafe();

偏移量



，获取value变量在对象内存中的偏移
部方法可以获取到变量value对其进行取值或赋值操作
et
ield("value"));
(ex); }
```

9

8

i是为了更进一步的保证线程安全。

他线程在之后的一小段时间内可以获取到旧值，有点类似

```
ıewValue);
```

e.compareAndSwapInt()方法

```
fset, newValue);
```

是value变量)

```
t, int update) {
lueOffset, expect, update);
```

```
fset, 1);
```

```
fset, -1);
```

```
fset, delta);
```

```
fset, 1) + 1;
```

```
fset, -1) - 1;
```

```
fset, delta) + delta;
```


9

8

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

关于 招聘 广告服务 网站地图

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

9

8

9

8

过Unsafe类中的CAS相关操作的方法实现的，这也同时

分析自增操作实现过程，其他方法自增

理一样。

getAndAddInt()方法实现了CAS操作，从而保证了线程

是Unsafe类中1.8新增的方法，源码如

```
set, int delta) {

, v + delta));
```

戍功为止，调用的是Unsafe类中的compareAndSwapInt

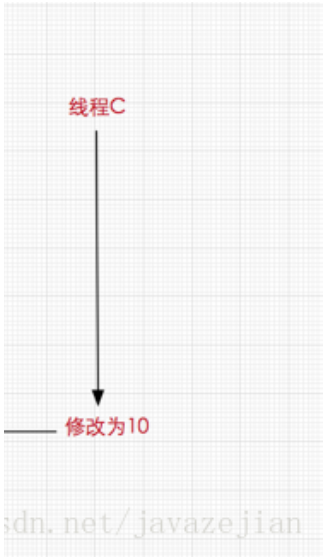
，上述源码分析是基于JDK1.8的，如果是1.8之前的方法，AtomicInteger源码实现有所不

cInteger实现该方法，
etAndAddInt方法即可

9

8

v，准备修改为新值U前，另外两个线程已连续修改了两次变量V的值，使得该值又恢复为旧值，这样的话，我们就无法正确判断这个变量是否已被修改过，如下图



发生了也不会造成什么问题，比如说我们对某个做加减法，不关心数字的过程，那么发生ABA问题也没啥关系。但是在某些情况下还是需要防止的，那么该如何解决呢？在Java中解决ABA问题，

用，在每次修改后，AtomicStampedReference

当AtomicStampedReference设

值时，对象值以及时间戳都必须满足期

写已被修改的窘境

9

8

饰的私有实例

()方法，

mpareAndSwapObject方法

交，

值和时间戳替换，也就避免了ABA的问题。

true和false两种切换状态，

题发生的概率。

这里不再介绍。到此，我们也明白了如果要完全杜绝ABA问题的发生，我们应该使用AtomicStampedReference原子类更新对象，而对于AtomicMarkableReference来说只能减少ABA问题的发

机会让当前想要获取锁的线程做几个空循环(这也是称为自旋的原因)，在经过若干次循环后，如果得到锁，

层面挂起，这种方式确实也是可以提升效率的。但问题是当线程越来越多竞争很激烈时，

般对于自旋锁有一定的次数限制，可能是50或者1(PU资源。


```
icReference<>();
```

9

8

当前线程，并将预期值设置为null。unlock()函数将

CAS操作也是通过不断的自循环(while循环)实现，

前线程。然后我们通过lock()和unlock来控制自旋锁的开启与关闭，注意这是一

于的值，但也是自旋锁的一种。

```
icReference<>();
```

9

8

当前线程，并将预期值设置为null。unlock()函数将 前线程。然后我们通过lock()和un 控制自旋锁的开启与关闭，注意这是一
CAS操作也是通过不断的自循环(while循环)实现， 于的值，但也是自旋锁的一种。

et(null, current)对应的签名是compareAndSet(E,V)还是compareAndSet(E,N)啊? (09-04 22:01 #8楼)

查看 8 条热评

13.4万

大的搜索，借此挖苦下百度，依靠百度什么都学习不到！ …

1.2万

大公司的日活动辄成百上千万。如何面对如此高的并发是…

大的场景，CAS机制中用了3个变量：内存值V，旧的预

4

机制存在以下问题： （1）在多线程竞争下，加锁、释放

1

rent包完全建立在CAS之上的，没有CAS就不...

1

决或者方法，也不要直接使用 jdk 提供的线程安全 的数据…

2593

指令，其作用是让CPU先进行比较两个值是否相等，然后…

37

AS 操作包含三个操作数 —— 内存位置（V）、预期原值…

1012

用中需要注意什么问题？ 这里以提问的方式引出话题，…

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

关于 招聘 广告服务 网站地图

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

9

8

ncurrency包的程序员。本文是本系列的第三篇文章，前.

1

rrent包的基础。 CAS有三个操作数，内存值M，旧的预其

are and set的缩写。JVM中的CAS操作利用了处理器提供

9

8

04-30

319

要百分之百完全控制才会去做，否则就认为这件事情一定会出问题；而乐观者的人生观则相反，凡事不管最终结果如何，他都会先尝试去做，大不了最后...

1696

(后面的章节还会谈到锁）。 锁机制存在以下问题： （1）在多线程竞争下，加锁、释放锁会导致比较多的上下文切换和...

1826

CAS的重要性。 CASCAS:Compare and Swap, 翻译成比较并交换。 java.util...

439

，Compare and Swap即比较并替换，设...

91

子性；如其名，主要操作是比较，设置。2.如何保证原子性 ...备份（备...

9

780

8

锁的线程挂起，等待持有锁的线程释放锁。 乐观锁：每次 ...为冲突失败就重试，直...

6655

ractQueuedSynchronizer)。 CAS(Compare And Swap) ...

个人资料

sylarji 关注

原创	粉丝	喜欢	评论
29	4	10	8

等级： 博客 2 访问： 8417

积分： 334 排名： 25万+

勋章：

最新文章

JAVA类加载机制和类加载器

JAVA之Executor线程池

浅谈JAVA之读写锁ReentrantReadWriteLock

JAVA之Semaphore信号量

java之ReentrantLock

个人分类

Jvm学习	1篇
学习笔记	37篇

归档

2018年3月	10篇
2018年2月	7篇
2018年1月	21篇

热门文章

JAVA中的CAS
阅读量：5929

JDK1.8之ConcurrentHashMap
阅读量：132

JDK1.8之LinkeList以及与ArrayList的区别
阅读量：122

快速失败fail-fast机制
阅读量：120

9

8

Git常用命令
阅读量：92

最新评论

JAVA中的CAS
stupidNameLimit：最后的SpinLock中，lock()方法里的compareAndSet(null, curren...

JAVA中的CAS
ytm15732625529：每一点自己都做了分析

JAVA中的CAS
programminging：能转载吗?

JAVA中的CAS
war2012：自旋锁这块还是不懂，哈哈

JAVA中的CAS
wuyongzhou：个人理解能力不足。。看起来头大

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

9

8