volatile保证内存可见性  
CAS算法保证原子性

## 1）CAS算法

* CAS（Compare-And-Swap，比较再交换）是一种硬件对并发的支持，针对多处理器操作而设计的处理器中的一种特殊指令，用于管理对共享数据的并发访问。
* CAS是一种无锁的非阻塞算法的实现。
* CAS包含了3个操作数：
* 需要读写的内存值V
* 进行比较的值A
* 拟写入的新值B
* 当且仅当V的值等于A时，CAS通过原子方式用新值B来更新V的值，否则不会执行任何操作。

## 2）原子变量

* java.util.concurrent类的小工具包，支持在单个变量上解除锁的线程安全编程。事实上，此包中的类可将volatile值、字段和数组元素的概念扩展到那些也提供原子条件更新操作的类。
* 类AtomicBoolean、AtomicInteger和AtomicReference的实例各自提供对相应类型单个变量的访问和更新。每个类也为该类型提供适当的实用工具方法。
* AtomicIntegerArray、AtomicLongArray和AtomicReferenceArray类进一步扩展了原子操作，对这些类型的数组提供了支持。这些类在为其数组元素提供volatile访问语义方面也引人注目，这对于普通数组来说是不受支持的。
* 核心方法：boolean compareAndSet(expectedValue, updateValue)
* java.util.concurrent.atomic包下提供一系列原子操作的常用类：
* AtomicBoolean、AtomicInteger、AtomicLong、AtomicReference
* AtomicIntegerArray、AtomicLongArray
* AtomicMarkableReference
* AtomicReferenceArray
* AtomicStampedReference



*/\*\*  
 \* i++的原子性问题：i++的操作实际上是3个步骤“读-改-写”  
 \*  
 \* int i = 10;  
 \* i = i++;  
 \*  
 \* int temp = i;  
 \* i = i + 1;  
 \* i = temp;  
 \*  
 \* 原子变量：jdk1.5后java.util.concurrent.atomic包下提供了常用的原子变量：  
 \* volatile保证内存可见性  
 \* CAS(compare-and-swap)算法保证原子性  
 \*/***public class** TestAtomic {  
  
 **public static void** main(String[] args) {  
 AtomicDeomo atomicDeomo = **new** AtomicDeomo();  
  
 **for** (**int** i = 0; i < 10; i++) {  
 **new** Thread(atomicDeomo).start();  
 }  
 }  
}  
  
**class** AtomicDeomo **implements** Runnable {  
  
 **private** AtomicInteger **serialNumber** = **new** AtomicInteger();  
  
 @Override  
 **public void** run() {  
  
 **try** {  
 Thread.*sleep*(200);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 System.***out***.println(Thread.*currentThread*().getName() + **":"** + getSerialNumber());  
 }  
  
 **public int** getSerialNumber() {  
 **return serialNumber**.getAndIncrement();  
 }  
}

原子操作：是指不会被线程调度机制打断的操作；这种操作一旦开始，就一直运行到结束，中间不会有任何context switch（切换到另一个线程）。

## 模拟CAS算法

*/\*\*  
 \* 根据CAS算法  
 \*/***public class** TestCAS {  
  
 **public static void** main(String[] args) {  
 **final** CompareAndSwap cas = **new** CompareAndSwap();  
  
 **for** (**int** i = 0; i < 10; i++) {  
 **new** Thread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **int** expectedValue = cas.get();  
 **boolean** b = cas.compareAndSet(expectedValue, (**int**) (Math.random() \* 101));  
 System.out.println(b);  
 }  
 }).start();  
 }  
 }  
}  
  
**class** CompareAndSwap {  
 **private int value**;  
  
 *//获取内存值* **public synchronized int** get() {  
 **return value**;  
 }  
  
 *//比较* **public synchronized int** compareAndSwap(**int** expectedValue, **int** newValue) {  
 **int** oldValue = **value**;  
  
 **if** (oldValue == expectedValue) {  
 **this**.**value** = newValue;  
 }  
 **return** oldValue;  
 }  
  
 *//设置* **public synchronized boolean** compareAndSet(**int** expectedValue, **int** newValue) {  
 **return** expectedValue == compareAndSwap(expectedValue, newValue);  
 }  
}