CAS翻译为比较并交换，在java.util.concurrent包中借助CAS实现了区别synorchinzed同步锁的一种乐观锁。

CAS的应用

CAS有三个操作数，**内存值V，旧的预期值A，要修改的新值B。**CAS指令执行时，当且仅当内存地址V的值与预期值A相等时，将内存地址V的值修改为B，否则就什么都不做。整个比较并替换的操作是一个原子操作。

CAS的全称为比较交换。执行函数：CAS(V,E,N) V表示要更新的变量，E表示预期值，N表示新值。

如果V值等于E值，则将V值设为N。如要更新的遍历不等于预期值，说明有其他线程做了更新，则当前线程什么都不做。

**是否存在 在判断V和E值是否相同的时候，切换了线程，更新了值，造成数据不一致？**

**不存在，**因为**CAS是一种系统原语，由若干条指令组成的，用于完成某个功能的一个过程，并且原语的执行必须是连续的。**

Unsafe 里的CAS操作相关

**CAS是一些CPU直接支持的指令**，也就是我们前面分析的无锁操作，在Java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。

并发包中的原子操作类（Atomic系列）

**挂起与恢复**

将一个线程进行挂起是通过park方法实现的，调用 park后，线程将一直阻塞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，其底层实现最终还是使用Unsafe.park()方法和Unsafe.unpark()方法

从jdk1.5开始提供了java.util.concurrent.atomic包，在该包中提供了许多基于CAS实现的原子操作类，主要分为以下4种类型：

1. 原子更新基本类型

**原子更新主要包括三个类：**

**AtmoicBoolean： AtomicInteger AtomicLong**

CAS虽然很高效的解决了原子操作问题，但是CAS仍然存在三大问题。

1. 循环时间长开销很大。
2. 只能保证一个共享变量的原子操作。
3. ABA问题。

**CAS 的ABA问题**

**假设这样一种场景：当一个线程执行CAS<V，E，U>操作，在获取到当前变量V，准备修改为最新值U前，另外两个线程连续改了变量v的值，使得该值又恢复为旧值。**

**解决方法：**

1. AtmociStampedReference类

它是一个带有时间戳的对象引用，在每次修改后，AtomicStampedReference不仅会设置新值而且还会记录更改的时间。当对象值以及时间戳都满足期望值才能写入成功。

1. AtomicMarkableReference则是将一个boolean值作是否有更改的标记，本质就是它的版本号只有两个，true和false，修改的时候在这两个版本号之间来回切换，这样做并不能解决ABA的问题，只是会降低ABA问题发生的几率而已；

**CAS 实现自旋锁**

**假设当前有两个线程1和2，线程1先调用lock，此时sign中的value，也就是旧值为null，所以sign .compareAndSet(null, current)返回true，并且将value更新为current，跳出循环；然后线程2来调用lock，显然线程1与线程2不相等，所以返回false，继续循环，直到线程1调用了unlock后将value置为null，线程2才能继续执行**

**public class SpinLock {**

**private AtomicReference<Thread> sign =new AtomicReference<>();**

**public void lock(){**

**Thread current = Thread.currentThread();**

**while(!sign .compareAndSet(null, current)){**

**}**

**}**

**public void unlock (){**

**Thread current = Thread.currentThread();**

**sign .compareAndSet(current, null);**

**}**

**}**