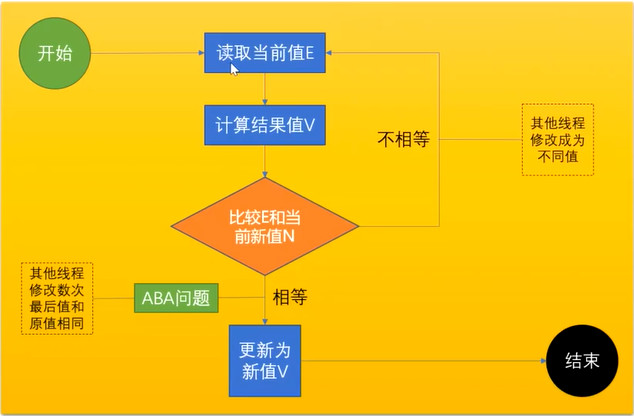
1.传统进行累加，使用i++,如果需要多个线程进行++,就需要进行加锁sychronized,但是这样效率就会比较低，所以在jdk1.5以后加入了一些原子类：AtomicInteger（无锁、自旋锁）。

AtomicInteger也能实现递增，而且不用加锁，他的实现就是运用了cas.

CAS:compare and swap(他可以保证在不加锁的情况下多个线程对一个值的更新)

他的实现建立的基础是在：Java中对赋值和数据从内存中读写的操作是原子性的。



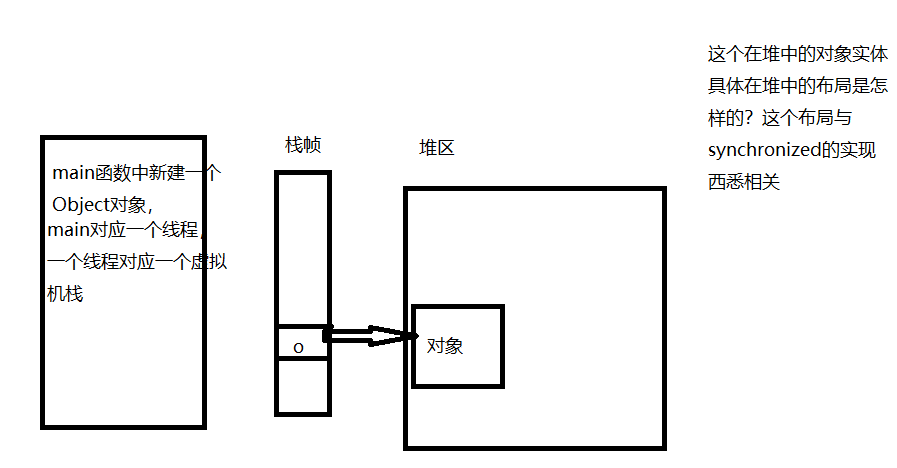
ABA问题：就是你要写回值时，先会看内存中变量值是否有变动，没有就写，有就重复过程，但是可能存在在你看到的没有改变值的现象只是假象（就像你和你女朋友分手后，一年之后又要去复合，你看到你女朋友还是那个样子，但是你能确定这一年中她没有经历过其他的人？这你是不清楚的。---从起点又回到起点，你以为没变，其实只是你不知道而已）,如果不影响，那就无所谓，有影响就需要进行处理。

如何解决ABA问题？：很简单，加个标志就行（加版本号），改一次修改版本号就行。具体到Java的实现是需要去使用AtomicStampedReference类。

**（问题1：这个版本号的值的读写会不会出现同样的问题？如果不会它又会怎么解决，这种解决的方案又可不可以反过来解决非原子性问题）**

CAS的底层实现：与synchronized、volatile基本一致。跟踪源码知道底层使用汇编实现，在硬件层面有一条汇编指令直接支持.而最终最终实现的指令是**lock cmpxchg指令.cmpxchg(直接与CAS对应)并不保证原子性,但是前面的lock保证了原子性**.(也就是说白了仅仅只是将底层的指令进行封装了而已)(lock指令在执行后面指令的时候,锁定了北桥芯片上的一个电信号,不采用锁总线的方式)

***JOL:Java Object Layout(Java对象布局)***

******

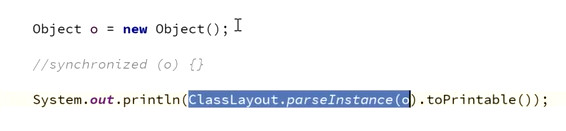
***这个对象的布局怎么能够看到?可以通过Java提供的JOL工具进行看.***

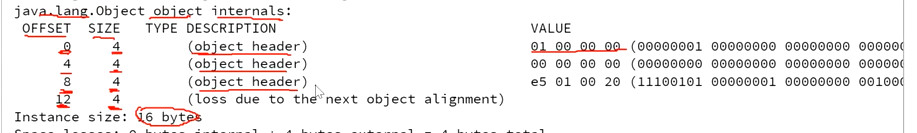
***导入工具包:***

***要么自己下jar包,要么就建立maven项目,配置文件导包.***



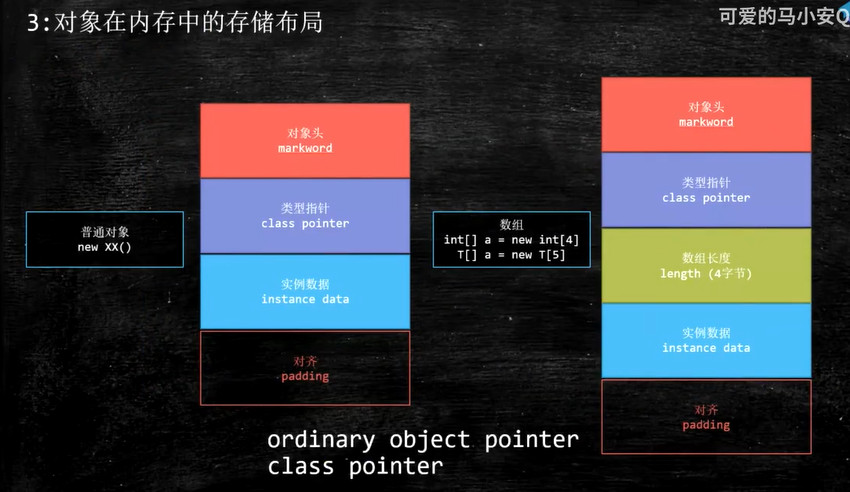
***打印出这个对象的布局:***



******

结果解释:从0字节开始,往后四个字节大小,是对象的头,它的值是---;从4字节开始,往后四个字节大小,是对象的头,它的值是---;从8字节开始,往后四个字节大小,是对象的头,它的值是---;从12字节开始,往后四个字节大小,(对齐字节数),它的值是---;（object没有成员变量）

最终实例的大小为16个字节



普通对象：

对象头：

Markword：

（类型指针）Class pointer：指明这个对象的类型。（因为现在的jvm已经是64位，所以它的指针长度应该是8个字节，但是Java会默认将其进行压缩成四个字节，即使是对象中的成员变量是一个引用/指针也会默认压缩为4个字节）

实例数据：对象的成员变量属性，比如含有一个int变量，那么这里就有4个字节，如果加再一个long变量就再加8个字节。

对齐（padding）：如果总字节数不是8的整数倍就会自动补为8的整数倍，因为自定义类读取内存中的值的时候按照总线的宽度来读，采用一次读取8个字节，所以如果对象所占用的总字节数是8的整数倍，这样会提高读取的速度。（就像）

**Synchronized的理解：因为你不能直接直接锁住代码块，而执行的代码的时候是顺序向下的，那么我直接在这个代码块前面加上上道门（那就是这个锁对象），你要过去就需要得到门的允许，这里要门允许就需要得到门的监视器，而监视器只有一个，所以，一旦被拿走，其他人就得不到门的认可，也就没有办法执行后面的代码。但是拿走了这个锁**

***现在仔细的分析上锁的过程：***



**从右往往左进行解析。**