原 java初、中、高级面试题——JVM、GC

2019年06月18日 18:34:10 在IT中穿梭旅行 阅读数 52 编辑

JVM

63.什么是Java虚拟机？为什么Java被称作是“平台无关的编程语言”？

考察点：JVM

参考回答：

虚拟机包括：系统虚拟机和程序虚拟机， 系统虚拟机如VMware 它完全对物理计算机仿真，提供一个可运行完整操作系统的软件平台。 程序虚拟机典型代表如java虚拟机，它专门为执行单个计算机程序而设计。

Java虚拟机是一台执行Java字节码的虚拟计算机，它拥有独特的运行机制。Java虚拟机中执行的指令我们成为java字节码指令。

Java 源文件被编译成能被 Java 虚拟机执行的字节码文件。 java要在应用平台上执行，需要先安装执行虚拟机，java虚拟机屏蔽了平台操作系统的底层硬件信息，抽取整理了各平台公共的处理硬件的接口提供给开发用户使用，java开发用户只要基于JVM开发java程序即可。

64.JVM基本概念

(1) 基本概念：

JVM是可运行Java代码的假想计算机 ，包括一套字节码指令集、一组寄存器、一个栈、一个垃圾回收，堆 和 一个存储方法域。JVM是运行在操作系统之上的，它与硬件没有直接的交互。

(2) 运行过程：

我们都知道Java源文件，通过编译器，能够生产相应的.Class文件，也就是字节码文件，而字节码文件又通过Java虚拟机中的解释器，编译成特定机器上的机器码 。

也就是如下：

① Java源文件—->编译器—->字节码文件

② 字节码文件—->JVM—->机器码

每一种平台的解释器是不同的，但是实现的虚拟机是相同的，这也就是Java为什么能够跨平台的原因了 ， 当一个程序从开始运行，这时虚拟机就开始实例化了，多个程序启动就会存在多个虚拟机实例。程序退出或者关闭，则虚拟机实例消亡，多个虚拟机实例之间数据不能共享。

(3) 三种JVM:

① Sun公司的HotSpot；

② BEA公司的JRockit；

③ IBM公司的J9 JVM；

在JDK1.7及其以前我们所使用的都是Sun公司的HotSpot，但由于Sun公司和BEA公司都被oracle收购，jdk1.8将采用Sun公司的HotSpot和BEA公司的JRockit两个JVM中精华形成jdk1.8的JVM。

请说明一下JAVA虚拟机的作用是什么?

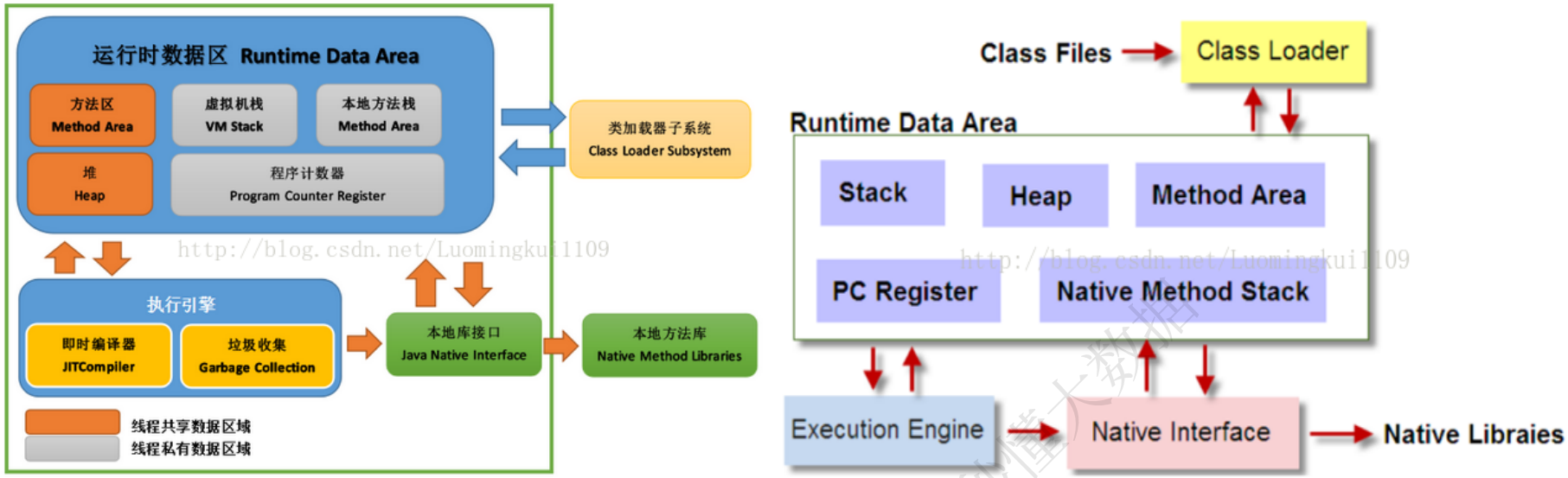
考察点：java虚拟机

参考回答:

解释运行字节码程序消除平台相关性。

jvm将java字节码解释为具体平台的具体指令。一般的高级语言如要在不同的平台上运行，至少需要编译成不同的目标代码。而引入JVM后，Java语言在不同平台上运行时不需要重新编译。Java语言使用模式Java虚拟机屏蔽了与具体平台相关的信息，使得Java语言编译程序只需生成在Java虚拟机上运行的目标代码（字节码），就可以在多种平台上不加修改地运行。Java虚拟机在执行字节码时，把字节码解释成具体平台上的机器指令执行。

65.JVM的体系结构



(1) Class Loader类加载器

负责加载 .class文件，class文件在文件开头有特定的文件标示，并且ClassLoader负责class文件的加载等，至于它是否可以运行，则由Execution Engine决定。

① 定位和导入二进制class文件

② 验证导入类的正确性

③ 为类分配初始化内存

④ 帮助解析符号引用.

(2) Native Interface本地接口:

本地接口的作用是融合不同的编程语言为Java所用，它的初衷是融合C/C++程序，Java诞生的时候C/C++横行的时候，要想立足，必须有调用C/C++程序，于是就在内存中专门开辟了一块区域处理标记为native的代码，它的具体作法是Native Method Stack中登记native方法，在Execution Engine执行时加载native libraies。

目前该方法使用的越来越少了，除非是与硬件有关的应用，比如通过Java程序驱动打印机，或者Java系统管理生产设备，在企业级应用中已经比较少见。

因为现在的异构领域间的通信很发达，比如可以使用Socket通信，也可以使用Web Service等。

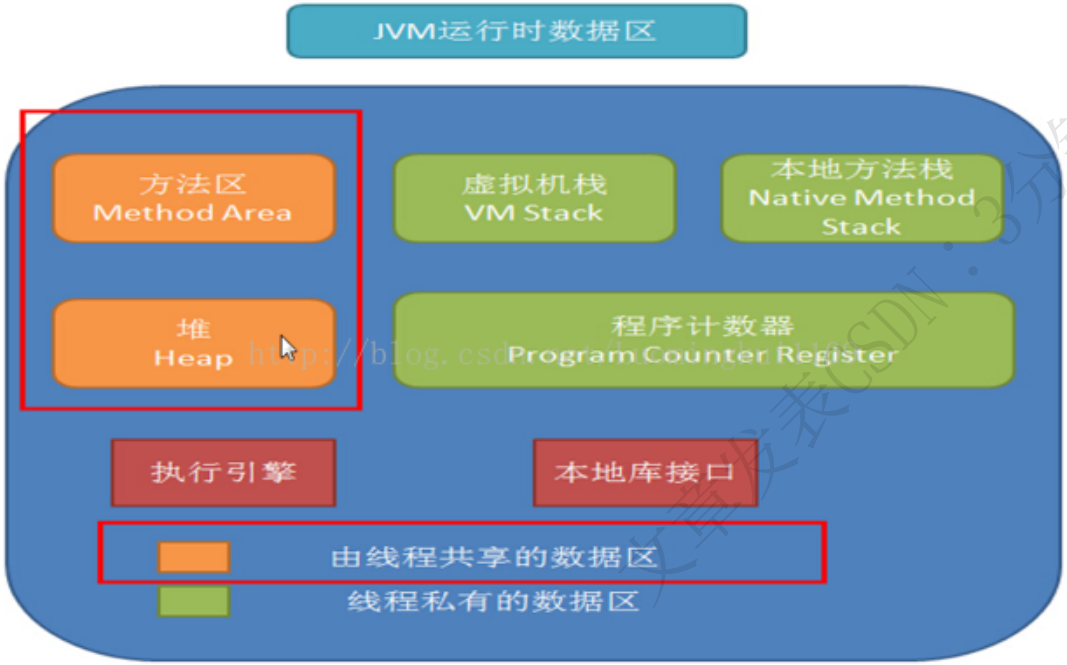
(3) Execution Engine 执行引擎：执行包在装载类的方法中的指令，也就是方法。

(4) Runtime data area 运行数据区:

虚拟机内存或者Jvm内存，冲整个计算机内存中开辟一块内存存储Jvm需要用到的对象，变量等，运行区数据有分很多小区，分别为：方法区，虚拟机栈，本地方法栈，堆，程序计数器。

66.JVM数据运行区详解（栈管运行，堆管存储）：

说明：JVM调优主要就是优化 Heap堆 和 Method Area 方法区。



1.程序计数器：是一个数据结构，用于保存当前正常执行的程序的内存地址。Java虚拟机的多线程就是通过线程轮流切换并分配处理器时间来实现的，为了线程切换后能恢复到正确的位置，每条线程都需要一个独立的程序计数器，互不影响，该区域为“线程私有”。

2.Java虚拟机栈：线程私有的，与线程生命周期相同，用于存储局部变量表，操作栈，方法返回值。局部变量表放着基本数据类型，还有对象的引用。

3.本地方法栈：跟虚拟机栈很像，不过它是为虚拟机使用到的Native方法服务。

4.方法区：各个线程共享的区域，储存虚拟机加载的类信息，常量，静态变量，编译后的代码。

5.Java堆：所有线程共享的一块内存区域，对象实例几乎都在这分配内存。

堆这块区域是JVM中最大的，应用的对象和数据都是存在这个区域，这块区域也是线程共享的，也是 gc 主要的回收区，一个 JVM 实例只存在一个堆类存，堆内存的大小是可以调节的。类加载器读取了类文件后，需要把类、方法、常变量放到堆内存中，以方便执行器执行，堆内存分为三部分：

① **新生代** 新生代有一个Eden区和两个survivor区，首先将对象放入Eden区，如果空间不足就向其中的一个survivor区上放，触发一次minor GC，如果仍然放不下就将存活的对象放入另一个survivor区中，然后清空Eden和之前的那个survivor区的内存。在某次GC过程中，如果发现仍然又放不下的对象，就将这些对象放入老年代内存里去。

② **老年代** 大对象以及长期存活的对象直接进入老年区。

③ **永久区** 永久存储区是一个常驻内存区域，用于存放JDK自身所携带的 Class,Interface 的元数据，也就是说它存储的是运行环境必须的类信息，被装载进此区域的数据是不会被垃圾回收器回收掉的，关闭 JVM 才会释放此区域所占用的内存。

如果出现java.lang.OutOfMemoryError: PermGen space，说明是Java虚拟机对永久代Perm内存设置不够。

67.jvm最大内存限制多少?

考察点：JVM

参考回答：

(1)堆内存分配

JVM初始分配的内存由-Xms指定，默认是物理内存的1/64；JVM最大分配的内存由-Xmx指定，默认是物理内存的1/4。默认空余堆内存小 于40%时，JVM就会增大堆直到-Xmx的最大限制；空余堆内存大于70%时，JVM会减少堆直到-Xms的最小限制。因此服务器一般设置-Xms、 -Xmx相等以避免在每次GC后调整堆的大小。

(2)非堆内存分配

JVM使用-XX:PermSize设置非堆内存初始值，默认是物理内存的1/64；由XX:MaxPermSize设置最大非堆内存的大小，默认是物理内存的1/4。

(3)JVM最大内存

首先JVM内存限制于实际的最大物理内存，假设物理内存无限大的话，JVM内存的最大值跟操作系统有很大的关系。简单的说就32位处理器虽然可控内存空间有4GB,但是具体的操作系统会给一个限制，这个限制一般是2GB-3GB（一般来说Windows系统下为1.5G-2G，Linux系 统下为2G-3G），而64bit以上的处理器就不会有限制了。

68.垃圾回收概念、原理

概念：将内存中、不会再被使用的对象进行回收。

原理：当对象被创建时，GC就开始监控这个对象的地址，大小及使用情况。通常，GC采用有向图的方式记录和管理堆(heap)中的所有对象。通过确定哪些对象时可达的，哪些是不可达的。当GC确定一些对象为”不可达”时，GC就有责任回收这些内存空间。

69.为什么要有GC?GC优点

GC是垃圾收集的意思（Gabage Collection）,内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，**Java提供的GC功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java语言没有提供释放已分配内存的显示操作方法。**

优点： 使得Java程序员在编写程序的时候不再需要考虑内存管理，垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。

70.简述一下GC算法

①GC（GarbageCollection 垃圾收集），GC的对象是堆空间和永久区

②GC算法包含：引用计数法，标记-清除算法，复制算法，标记-压缩算法，分代收集算法

1.引用计数法：引用计数器的实现很简单，对于一个对象A，只要有任何一个对象引用了A，则A的引用计数器就加1，当引用失效时，引用计数器就减1。只要对象A的引用计数器的值为0，则对象A就不可能再被使用。

缺点：1无法处理循环引用问情况。**会造成内存泄漏** 2.对系统性能产生影响。

2.标记-清除算法：将垃圾回收分为两个阶段：标记阶段和清除阶段，首先标记出所有需要回收的对象，在标记完成后统一回收所有被标记的对象。

缺点：1 效率问题，2，空间问题，标记清除之后会产生大量不连续的内存碎片，空间碎片太多会导致以后程序在运行过程中需要分配较大对象时，无法找到足够的连续内存而提前触发另一次垃圾收集动作。

3.复制算法：将可用内存按容量划分为大小相等的两块，每次只试用其中的一块，当这一块内存用完时，将存活的对象复制到另外一块内存上面，然后清除使用内存中的所有对象。 **适用于初生代。**

4.标记压缩算法： 首先标记出所有需要回收的对象，然后让所有存活的对象都向一端移动，然后清理掉端边界以外的内存。 **适用于老年代**

5.分代收集算法： 初生代使用复制算法，老年代使用标记压缩算法。

71.请问java中内存泄漏是什么意思？什么场景下会出现内存泄漏的情况?

Java中的内存泄露，就是：不再被使用的对象的内存不能被回收，就是内存泄露。

例如：有对象A和对象B，对象A中含有对象B的引用，对象B中含有对象A的引用。对此，对象A和对象B的引用计数器都不为0，但是在系统中，却不存在任何第三个对象引用A和B，所以A和B应该被回收，但是又因为A和B 互相引用，所以垃圾回收器无法识别，引起内存泄漏。

71请说明一下eden区和survial区的含义以及工作原理?

考察点：JVM

参考回答：

目前主流的虚拟机实现都采用了分代收集的思想，把整个堆区划分为新生代和老年代；新生代又被划分成Eden 空间、 From Survivor 和 To Survivor 三块区域。

我们把Eden : From Survivor : To Survivor 空间大小设成 8 : 1 : 1 ，对象总是在 Eden 区出生， From Survivor 保存当前的幸存对象， To Survivor 为空。一次 gc 发生后： 1）Eden 区活着的对象 + From Survivor 存储的对象被复制到 To Survivor ；
2) 清空 Eden 和 From Survivor ； 3) 颠倒 From Survivor 和 To Survivor 的逻辑关系： From 变 To ， To 变 From 。可以看出，只有在 Eden 空间快满的时候才会触发 Minor GC 。而 Eden 空间占新生代的绝大部分，所以 Minor GC 的频率得以降低。当然，使用两个 Survivor 这种方式我们也付出了一定的代价，如 10% 的空间浪费、复制对象的开销等。

72 Minor GC ， Full GC 触发条件是什么?

- 对老年代GC称为Major GC；
- 而Full GC是对整个堆来说的；

在最近几个版本的JDK里默认包括了对永生带即方法区的回收（JDK8中无永生带了），出现Full GC的时候经常伴随至少一次的Minor GC,但非绝对的。Major GC的速度一般会比Minor GC慢10倍以上。下边看看有那种情况触发JVM进行Full GC及应对策略。

Minor GC触发条件：
当Eden区满时，触发Minor GC。

Full GC触发条件：

（1）System.gc()方法的调用

（2）老年代空间不足

（3）方法区空间不足

（4）通过Minor GC后进入老年代的平均大小大于老年代的可用内存

（5）由Eden区、From Space区向To Space区复制时，对象大小大于To Space可用内存，则把该对象转存到老年代，且老年代的可用内存小于该对象大小

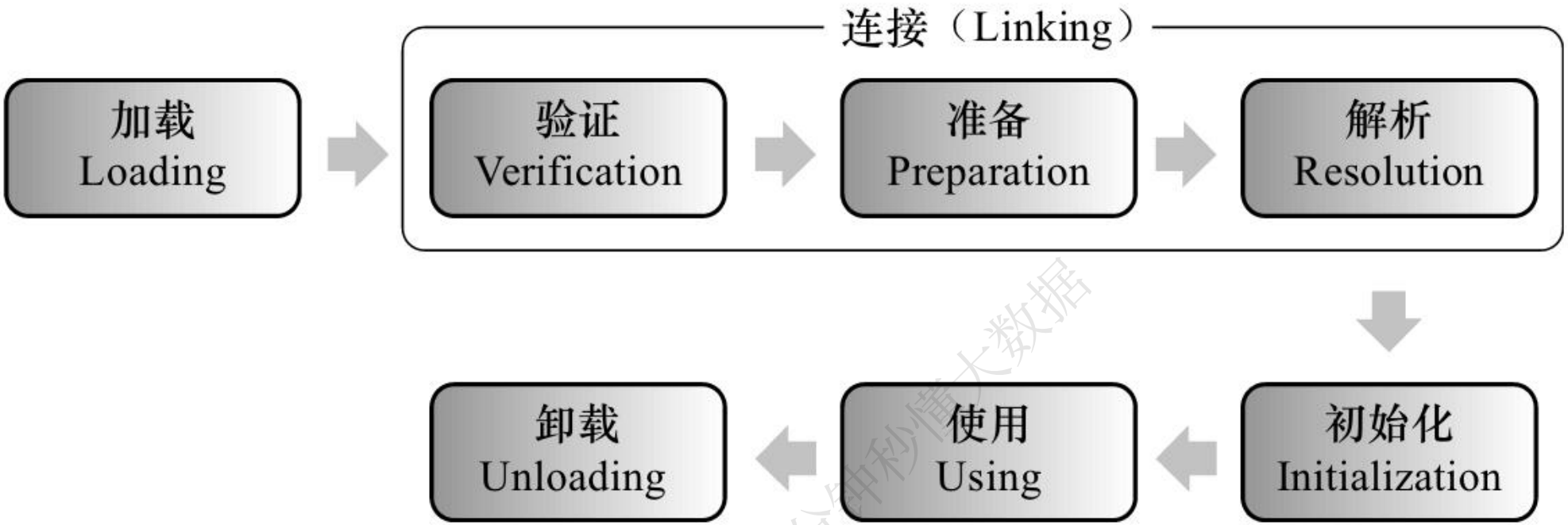
72 Minor GC 和Full GC有什么不一样吗？

新生代GC（Minor GC）:指发生在新生代的垃圾收集动作，因为java对象大多都具备朝生夕灭的特性，所以MinorGC非常频繁，一般回收速度也比较快。

老年代GC（Major GC/Full GC）:指发生在老年代的GC，出现了MajorGC,经常会伴随至少一次的MinorGC。MajorGC的速度一般会比MinorGC慢10倍以上。

73.描述一下Class类装载过程。

JVM类装载机制分为五个部分：加载，验证，准备，解析，初始化



Class只有在必须要使用的时候才会被装载，java虚拟机不会无条件地装载Class类型。java虚拟机规定，一个类或者接口在初次使用前，必须要进行初始化。 这里的"使用"是指主动使用，主动使用只有以下几种情况。

1. 当创建一个类的实例时，比如使用new关键字，或者通过反射，克隆、反序列化
2. 当调用类的静态方法时，即当使用了字节码invokestatic指令。
3. 当使用类或接口的静态字段时（final常量除外），比如使用getstatic或者putstatic指令。
4. 当使用java.lang.reflect包中的方法反射类的方法时。
5. 当初始化子类时，要求初始化父类。
6. 作为启动虚拟机，含有main()方法的那个类。

装载的顺序：

加载——验证——准备——解析——初始化

1.**装载**：将Java二进制代码导入jvm中，生成Class文件。
2.**连接**：a）**校验**：检查载入Class文件数据的正确性 b）**准备**：给类的静态变量分配存储空间 c）**解析**：**将符号引用转成直接引用**
3：**初始化**：对类的静态变量，静态方法和静态代码块执行初始化工作。

解析：

解析阶段是指虚拟机将常量池中的符号引用替换为直接引用的过程。符号引用就是class文件中的：

CONSTANT_Class_info

CONSTANT_Field_info

CONSTANT_Method_info

等类型的常量。

符号引用与虚拟机实现的布局无关，引用的目标并不一定要已经加载到内存中。各种虚拟机实现的内存布局可以各不相同，但是它们能接受的符号引用必须是一致的，因为符号引用的字面量形式明确定义在Java虚拟机规范的Class文件格式中。

直接引用可以是指向目标的指针，相对偏移量或是一个能间接定位到目标的句柄。如果有了直接引用，那引用的目标必定已经在内存中存在。

双亲委派模型：类加载器收到类加载请求，首先将请求委派给父类加载器完成
用户自定义加载器->应用程序加载器->扩展类加载器->启动类加载器。

74.理解CAS

CAS是一种基于比较并交换(Compare And Swap)CAS算法的无锁并发控制方法。

与锁的实现相比较，无锁算法的设计和实现都要复杂得多，但由于其非阻塞性，他对死锁问题天生免疫，并且，线程间相互影响也远远比基于锁的方式要小。更重要的是，使用无锁的方式完全没有锁竞争带来的系统开销，也没有线程间频繁调度带来的开销，因此它比基于锁的方式拥有更优越的性能。

CAS（Compare And Swap） 无锁算法：
CAS是乐观锁技术，当多个线程尝试使用CAS同时更新同一个变量时，只有其中一个线程能更新变量的值，而其它线程都失败，失败的线程并不会被挂起，而是被告知这次竞争中失败，并可以再次尝试。CAS有3个操作数，内存值V，旧的预期值A，要修改的新值B。当且仅当预期值A和内存值V相同时，将内存值V修改为B，否则什么都不做。

```
int compare_and_swap (int* reg, int oldval, int newval)
{
  ATOMIC();
  int old_reg_val = *reg;
  if (old_reg_val == oldval)
     *reg = newval;
  END_ATOMIC();
  return old_reg_val;
}
```

75 原子操作

为了让CAS操作被java应用程序充分使用，在JDK的java.util.conurrent.atomic包下，有一组使用无锁算法实现的原子操作类，主要有AtomicInteger，AtomicIntegerArray、AtomicLong、AtomicLongArray和AtomicReference等。他们分别封装了对整数，整数数组、长整型、长整型数组、和普通对象的多线程安全操作。

Java中的LongAdder和AtomicLong有什么区别?

JDK1.8引入了LongAdder类。CAS机制就是，在一个死循环内，不断尝试修改目标值，直到修改成功。如果竞争不激烈，那么修改成功的概率就很高，否则，修改失败的的概率就很高，在大量修改失败时，这些原子操作就会进行多次循环尝试，因此性能就会受到影响。 结合ConcurrentHashMap的实现思想，应该可以想到对一种传统AtomicInteger等原子类的改进思路。虽然CAS操作没有锁，但是像减少粒度这种分离热点的思想依然可以使用。**将AtomicInteger的内部核心数据value分离成一个数组，每个线程访问时，通过哈希等算法映射到其中一个数字进行计数，而最终的计数结果，则为这个数组的求和累加。热点数据value被分离成多个单元cell，每个cell独自维护内部的值，当前对象的实际值由所有的cell累计合成，这样热点就进行了有效的分离，提高了并行度。**

LongAdder与类AtomicLong类的区别在于高并发时前者将对单一变量的CAS分散操作为对数组cells中多个元素的CAS操作，取值时进行求和;在而并发较低时仅对base变量进行CAS操作，与AtomicLong类原理相同。不得不说这种分布式的设计还是很巧妙的。