Java虚拟机：Java内存区域及对象

IT哈哈 今天

几个计算机的概念

为以后写文章考虑，也为巩固自己的知识和一些基本概念，这里要理清楚几个计算机中的概念。

1、计算机存储单位

从小到大依次为位Bit、字节Byte、千字节KB、兆M、千兆GB、TB，相邻单位之间都是1024倍，1024为2的10次方，即：

- 1Byte = 8bit
- 1K = 1024Byte
- 1M = 1024K
- 1G = 1024M
- 1T = 1024G

2、计算机存储元件

寄存器：中央处理器CPU的一部分，是计算机中读写速度最快的存储元件，但是容量很少

内存：属于独立的一个部件，是和CPU沟通的桥梁，用于存放CPU中的运算数据以及与外部存储器交换的数据。尽管在今天，对内存的读写速度已经很快了，但是由于寄存器是在CPU上的，所以对于内存的读写速度和对于寄存器的读写速度上还是有几个数量级的差距。但是没办法，对于内存的读写I/O操作是很难消除的，寄存器数量有限，不可能通过寄存器来完成所有的运算任务

3、内核空间和用户空间

连接内存和寄存器的是地址总线，地址总线的宽度影响了物理地址的索引范围，因为总线宽度决定了处理器一次可以从寄存器或内存中获取多少个Bit，同时也决定了处理器最大可以寻址的地址空间。比如32位CPU的系统，可寻址范围为0x00000000~0xFFFFFFFF，即232=4294967296个内存位置，每个内存位置1个字节，即32位CPU系统可以有4GB的内存空间。不过应用程序是不可以完全使用这些地址空间的，因为这些地址空间被划分为了内核空间和用户空间，程序只能使用用户空间的内存。内核空间主要是指操作系统运行时所使用的用于程序调度、虚拟内存的使用或者链接硬

件资源的程序逻辑。区分内核空间和用户空间的目的主要是从系统的稳定性的角度考虑的。Windows 32操作系统默认内核空间和用户空间的比例是1:1，即2G内核空间、2G内存空间，32位Linux系统中默认比例则是1:3，即1G内核空间，3G内存空间。

4、字长

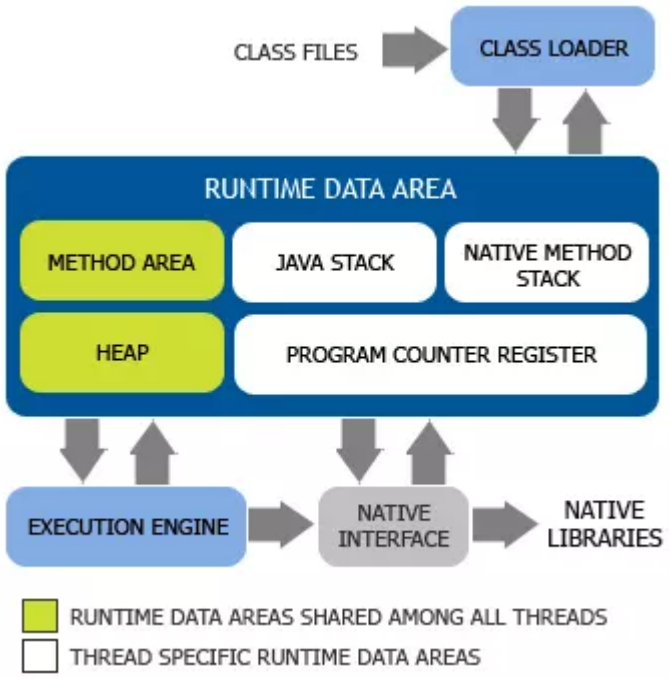
CPU的主要技术指标之一，指的是CPU一次能并行处理二进制的位数（Bit）。通常称处理字长为8位数据的CPU为8位CPU，32位CPU就是在同一时间内处理字长为32位的二进制数据。不过目前虽然CPU大多是64位的，但还是以32位字长运行

前言

说到Java内存区域，可能很多人第一反应是“堆栈”。首先堆栈不是一个概念，而是两个概念，堆和栈是两块不同的内存区域，简单理解的话，堆是用来存放对象而栈是用来执行程序的。其次，堆内存和栈内存的这种划分方式比较粗糙，这种划分方式只能说明大多数程序员最关注的、与对象内存分配关系最密切的内存区域是这两块，Java内存区域的划分实际上远比这复杂。对于Java程序员来说，在虚拟机自动内存管理机制的帮助下，不再需要为每一个new操作去配对delete/free代码，不容易出现内存泄露和内存溢出问题。但是，也正是因为Java把内存控制权交给了虚拟机，一旦出现内存泄露和内存溢出的问题，就难以排查，因此一个好的Java程序员应该去了解虚拟机的内存区域以及会引起内存泄露和内存溢出的场景。

运行时数据区域

Java虚拟机（JVM）内部定义了程序在运行时需要使用到的内存区域，从http://images.blogjava.net/blogjava_net/nkjava/jvmstructure.png拷贝一张图下来



之所以要划分这么多区域出来是因为这些区域都有自己的用途，以及创建和销毁的时间。有些区域随着虚拟机进程的启动而存在，有的区域则依赖用户线程的启动和结束而销毁和建立。图中绿色部分就是所有线程之间共享的内存区域，而白色部分则是线程运行时独有的数据区域，从这个分类角度来看一下这几个数据区。

1、线程独有的内存区域

（1）PROGRAM COUNTER REGISTER，程序计数器

这块内存区域很小，它是当前线程所执行的字节码的行号指示器，字节码解释器通过改变这个计数器的值来选取下一条需要执行的字节码指令。Java方法这个计数器才有值，如果执行的是一个Native方法，那这个计数器是空的。

（2）JAVA STACK，虚拟机栈

生命周期和线程相同。每个方法执行的同时都会创建一个栈帧，用于存储局部变量表、操作数栈、动态链接、方法出口等信息，每一个方法从调用直至执行完毕的过程，就对应着一个栈帧在虚拟机中入栈到出栈的过程。栈的大小和具体JVM的实现有关，通常在256K~756K之间。

（3）NATIVE METHOD STACK，方法栈

和虚拟机栈起的作用一样，只不过方法栈为虚拟机使用到的Native方法服务。虚拟机规范并没有对这个区域有什么强制规定，因此我们使用的HotSpot虚拟机，就干脆没有这块区域了，它和虚拟机栈是一起的。

2、线程间共享的内存区域

（1）HEAP，堆

大多数应用，堆都是Java虚拟机所管理的内存中最大的一块，它在虚拟机启动时创建，此内存唯一的目的就是存放对象实例。由于现在垃圾收集器采用的基本都是分代收集算法，所以堆还可以细分为新生代和老年代，再细致一点还有Eden区、From Survivior区、To Survivor区，这个后面都会讲到的。

（2）METHOD AREA，方法区

这块区域用于存储虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据，虚拟机规范是把这块区域描述为堆的一个逻辑部分的，但实际它应该是要和堆区分开的。从上面提到的分代收集算法的角度看，HotSpot中，方法区≈永久代。不过JDK 7之后，我们使用的HotSpot应该就没有永久代这个概念了，会采用Native Memory来实现方法区的规划了。

（3）RUNTIME CONSTANT POOL，运行时常量池

上面的图中没有画出来，因为它是方法区的一部分。Class文件中除了有类的版本信息、字段、方法、接口等描述信息外，还有一项信息就是常量池，用于存放编译期间生成的各种字面量和符号引用，这部分内容将在类加载后进入方法区的运行时常量池中，另外翻译出来的直接引用也会存储在这个区域中。这个区域另外一个特点就是动态性，Java并不要求常量就一定要在编译期间才能产生，运行期间也可以在这个区域放入新的内容，String.intern()方法就是这个特性的应用。

3、直接内存

想想还是把这块加上。直接内存并不是虚拟机运行时数据区的一部分，也不是Java虚拟机规范中定义的内存区域。但是这部分内存也被频繁地使用，而且也可能导致内存溢出问题。JDK1.4中新增加了NIO，引入了一种基于通道与缓冲区的I/O方式，它可以使用Native函数库直接分配堆外内存，然后通过一个存储在Java堆中的DirectByteBuffer对象作为这块内存的引用进行操作。这样能在一些场景中显著提高性能，因为避免了在Java堆和

Native堆中来回复制数据。显然，本机直接内存的分配不会受到Java堆大小的限制，但是，既然是内存，肯定还是会受到本机总内存（包括RAM、SWAP区）大小以及处理器寻址空间的限制。

对象创建

Java是一门面向对象的语言，Java程序运行过程中无时无刻都有对象被创建出来。在语言层面上，创建对象（克隆、反序列化）就是一个new关键字而已，但是虚拟机层面上却不是如此。看一下在虚拟机层面上创建对象的步骤：

1、虚拟机遇到一条new指令，首先去检查这个指令的参数能否在常量池中定位到一个类的符号引用，并且检查这个符号引用代表的类是否已经被加载、解析和初始化。如果没有，那么必须先执行类的初始化过程。

2、类加载检查通过后，虚拟机为新生对象分配内存。对象所需内存大小在类加载完成后便可以完全确定，为对象分配空间无非就是从Java堆中划分出一块确定大小的内存而已。这个地方会有两个问题：

（1）如果内存是规整的，那么虚拟机将采用的是指针碰撞法来为对象分配内存。意思是所有用过的内存在一边，空闲的内存在另外一边，中间放着一个指针作为分界点的指示器，分配内存就仅仅是把指针向空闲那边挪动一段与对象大小相等的距离罢了。如果垃圾收集器选择的是Serial、ParNew这种基于压缩算法的，虚拟机采用这种分配方式。

（2）如果内存不是规整的，已使用的内存和未使用的内存相互交错，那么虚拟机将采用的是空闲列表法来为对象分配内存。意思是虚拟机维护了一个列表，记录上哪些内存块是可用的，再分配的时候从列表中找到一块足够大的空间划分给对象实例，并更新列表上的内容。如果垃圾收集器选择的是CMS这种基于标记-清除算法的，虚拟机采用这种分配方式。

另外一个问题及时保证new对象时候的线程安全性。因为可能出现虚拟机正在给对象A分配内存，指针还没有来得及修改，对象B又同时使用了原来的指针来分配内存的情况。虚拟机采用了CAS配上失败重试的方式保证更新更新操作的原子性和TLAB两种方式来解决这个问题。

3、内存分配结束，虚拟机将分配到的内存空间都初始化为零值（不包括对象头）。这一步保证了对象的实例字段在Java代码中可以不用赋初始值就可以直接使用，程序能访问到这些字段的数据类型所对应的零值。

4、对对象进行必要的设置，例如这个对象是哪个类的实例、如何才能找到类的元数据信息、对象的哈希码、对象的GC分代年龄等信息，这些信息存放在对象的对象头中。

5、执行<init>方法，把对象按照程序员的意愿进行初始化，这样一个真正可用的对象才算完全产生出来。

以上这部分内容，如果有下载OpenJDK的源代码的话，可以通过参考hotspot/src/share/vm/interpreter/bytecodeInterpreter.cpp文件，从1939行开始。1939行的代码是CASE(_new):{...}，意思是当代码中遇见new这个关键字，虚拟机做的事情。实际虚拟机可能并不是执行的这段代码，但是通过这段代码来了解new对象的时候虚拟机的运作过程基本上是没问题的。

对象定位方式

建立对象是为了使用对象，Java程序需要通过栈上的reference（引用）数据来操作堆上的具体对象。比如我们写了一句
Object obj = new Object()
而new Object()之后其实有两部分内容，一部分是类数据（比如代表类的Class对象）、一部分是实例数据由于reference在Java虚拟机规范中只是一个指向对象new Object()的引用obj，并没有规定obj应该通过何种方式去定位、访问堆中对象的具体位置，所以对象访问方式也是取决于虚拟机而定的。主流方式有两种：

1、句柄访问。Java堆中划分出一块句柄池，obj指向的是对象的句柄地址，句柄中则包含了类数据的地址和实例数据的地址

2、指针访问。对象中存储所有的实例数据和类数据的地址，obj指向的是这个对象

HotSpot虚拟机采用的是后者，不过前者的对象访问方式也是十分常见的。

> 来源：http://www.cnblogs.com/xrq730/p/4827590.html

推荐阅读：

Java性能优化的50个细节（珍藏版）

Java性能优化之编程技巧总结

基于Spring Boot和Spring Cloud实现微服务架构

细思极恐-你真的会写Java吗?

Java 面试知识点解析——JVM篇

高级架构进阶之HashMap源码就该这么学(二)-get方法，remove方法