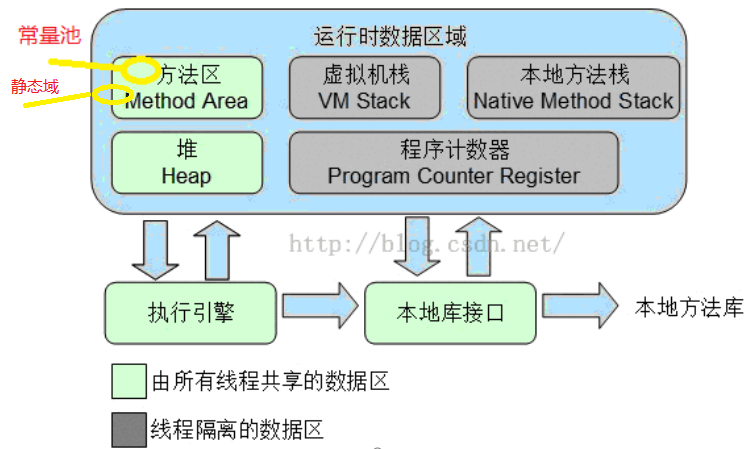
java内存管理（堆、栈、方法区）：https://blog.csdn.net/Liveor\_Die/article/details/77895631

# 内存管理

## 概述

Java虚拟机在执行Java程序的过程中会把它所管理的内存划分为若干不同的数据区域，这些区域都有各自的用途以及创建和销毁的时间。Java虚拟机所管理的内存将会包括以下几个运行时数据区域，如下图所示：



了解栈内存（Stack）和堆内存（Heap）和方法区（Method Area）这三部分

## 内存分配

栈： 存储局部变量（定义在方法声明上和方法中的变量） 值

堆： 存储new出来的数组或对象 地址

方法区： 面向对象部分讲解

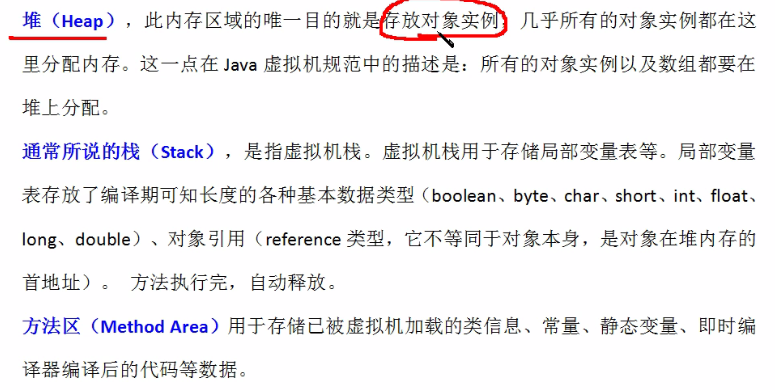
本地方法区： 和系统相关

寄存器： 给CPU使用

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 变量 | | 常量 | 其他 |
| 栈 | 局部变量 | 基本数据类型  对象数组引用 |  |  |
| 堆 | 实例变量 | 对象数组实例 |  |  |
| 方法区 | 类变量 |  | 常量 | 类信息，  即使编译代码 |

# 堆栈，方法区

## 总述



## 堆栈

### Java堆

堆内存用来存放由new创建的对象实例和数组。（重点）

　　Java堆是所有线程共享的一块内存区域，在虚拟机启动时创建，此内存区域的唯一目的就是存放对象实例 。

　　Java堆是垃圾收集器管理的主要区域。由于现在收集器基本采用分代回收算法，所以Java堆还可细分为：新生代和老年代。从内存分配的角度来看，线程共享的Java堆中可能划分出多个线程私有的分配缓冲区(TLAB)。

　　Java堆可以处于物理上不连续的内存空间，只要逻辑上连续的即可。在实现上，既可以实现固定大小的，也可以是扩展的。

　　如果堆中没有内存完成实例分配，并且堆也无法完成扩展时，将会抛出OutOfMemoryError异常。

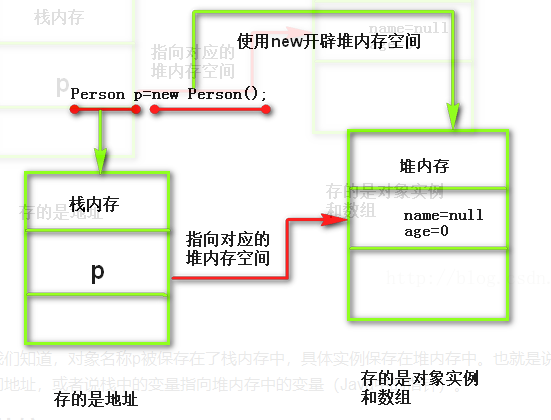
### Java栈

　　 在栈内存中保存的是堆内存空间的访问地址，或者说栈中的变量指向堆内存中的变量（Java中的指针）（重点）。

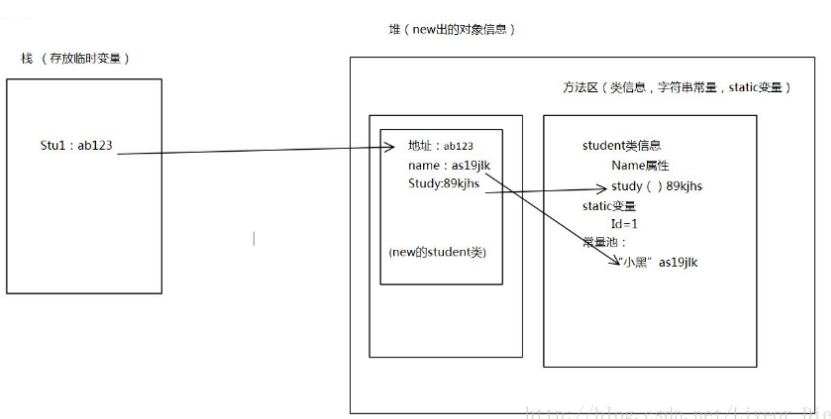
　　Java栈是Java方法执行的内存模型每个方法在执行的同时都会创建一个栈帧的用于存储局部变量表、操作数栈、动态链接、方法出口等信息。每个方法从调用直至执行完成的过程就对应着一个栈帧在虚拟机中入栈和出栈的过程。

### 堆和栈的联系

当在堆中产生了一个数组或者对象时，可以在栈中定义一个特殊的变量，让栈中的这个变量的取值等于数组或对象在堆内存中的首地址，栈中的这个变量就成了数组或对象的引用变量，以后就可以在程序中使用栈中的引用变量来访问堆中的数组或者对象，引用变量就相当于是为数组或者对象起的一个名称。引用变量是普通的变量，定义时在栈中分配，引用变量在程序运行到其作用域之外后被释放。而数组和对象本身在堆中分配，即使程序运行到使用new产生数组或者对象的语句所在的代码块之外，数组和对象本身占据的内存不会被释放，数组和对象在没有引用变量指向它的时候，才变为垃圾，不能在被使用，但仍然占据内存空间不放，在随后的一个不确定的时间被垃圾回收器收走（释放掉）。例如：



由上图我们知道，对象名称p被保存在了栈内存中，具体实例保存在堆内存中。也就是说，在栈内存中保存的是堆内存空间的访问地址，或者说栈中的变量指向堆内存中的变量（Java中的指针）。



### 堆和栈的比较

从堆和栈的功能和作用来通俗的比较,堆主要用来存放对象的，栈主要是用来执行程序的.而这种不同又主要是由于堆和栈的特点决定的:

　　在编程中，例如C/C++中，所有的方法调用都是通过栈来进行的,所有的局部变量,形式参数都是从栈中分配内存空间的。实际上也不是什么分配,只是从栈顶向上用就行,就好像工厂中的传送带一样,Stack Pointer会自动指引你到放东西的位置,你所要做的只是把东西放下来就行.退出函数的时候，修改栈指针就可以把栈中的内容销毁.这样的模式速度最快, 当然要用来运行程序了.需要注意的是,在分配的时候,比如为一个即将要调用的程序模块分配数据区时,应事先知道这个数据区的大小,也就说是虽然分配是在程序运行时进行的,但是分配的大小多少是确定的,不变的,而这个”大小多少”是在编译时确定的,不是在运行时.

　　堆是应用程序在运行的时候请求操作系统分配给自己内存，由于从操作系统管理的内存分配,所以在分配和销毁时都要占用时间，因此用堆的效率非常低.但是堆的优点在于,编译器不必知道要从堆里分配多少存储空间，也不必知道存储的数据要在堆里停留多长的时间,因此,用堆保存数据时会得到更大的灵活性。事实上,面向对象的多态性,堆内存分配是必不可少的,因为多态变量所需的存储空间只有在运行时创建了对象之后才能确定.在C++中，要求创建一个对象时，只需用 new命令编制相关的代码即可。执行这些代码时，会在堆里自动进行数据的保存.当然，为达到这种灵活性，必然会付出一定的代价:在堆里分配存储空间时会花掉更长的时间。

## 方法区

### 方法区

方法区是各个线程共享的内存区域，它用于存储已被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 （重点）。

　　相对而言，垃圾收集行为在这个区域比较少出现，但并非数据进了方法区就永久的存在了，这个区域的内存回收目标主要是针对常量池的回收和对类型的卸载，

　　当方法区无法满足内存分配需要时，将抛出OutOfMemoryError异常。

### **运行时常量池**

是方法区的一部分，它用于存放编译期生成的各种字面量和符号引用。

### 静态区

# 了解

## 程序计数器（了解）

程序计数器，可以看做是当前线程所执行的字节码的行号指示器。在虚拟机的概念模型里，字节码解释器工作就是通过改变程序计数器的值来选择下一条需要执行的字节码指令，分支、循环、跳转、异常处理、线程恢复等基础功能都要依赖这个计数器来完成

## Java虚拟机栈（了解）

Java虚拟机栈也是线程私有的 ，它的生命周期与线程相同。虚拟机栈描述的是Java方法执行的内存模型：每个方法在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、动态链表、方法出口信息等。每一个方法从调用直至执行完成的过程，就对应着一个栈帧在虚拟机栈中入栈到出栈的过程。

局部变量表中存放了编译器可知的各种基本数据类型(boolean、byte、char、short、int、float、long、double)、对象引用和returnAddress类型(指向了一条字节码指令的地址)。

如果扩展时无法申请到足够的内存，就会抛出OutOfMemoryError异常。

## 本地方法栈（了解）

本地方法栈与虚拟机的作用相似，不同之处在于虚拟机栈为虚拟机执行的Java方法服务，而本地方法栈则为虚拟机使用到的Native方法服务。有的虚拟机直接把本地方法栈和虚拟机栈合二为一。

会抛出stackOverflowError和OutOfMemoryError异常。