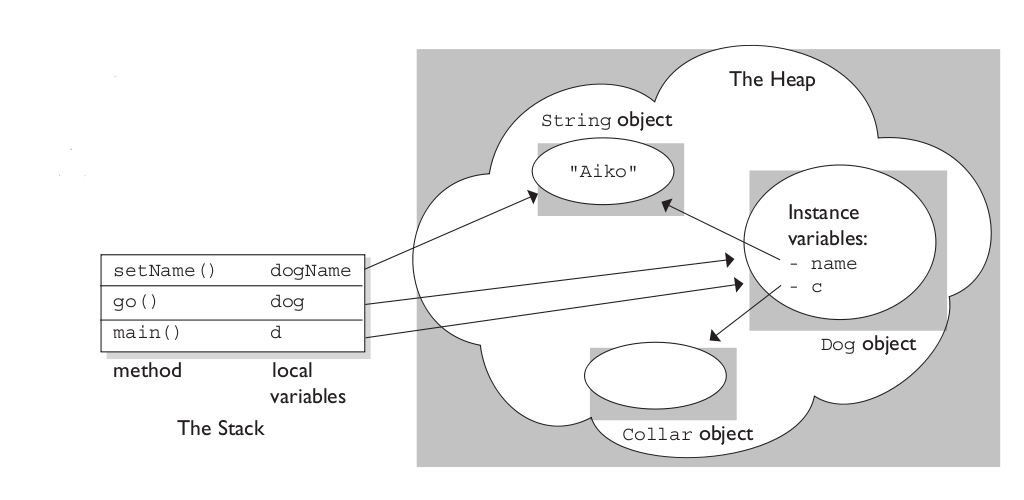
堆和栈都在内存（CPU cache / RAM / swap）中

栈：存取速度比堆要快（因为栈的本质是指针移动，操作比堆简单，并且由于栈更易被反复使用，容易被放到cpu cache内，另外**堆在线程**（jvm只有一个堆）之间是互通的，所以加锁会导致慢）。由操作系统自动分配释放。存在栈中的**数据大小与生存期**必须是确定的。操作方式类似于数据结构中的栈。**存在栈中的数据可以共享（堆不可以）。**在栈式存储方案中,程序对数据区的需求**在编译时确定**,只有到运行的时候才分配,但是规定在运行中**进入一个程序模块时为其分配内存。**当进入一个function时，栈顶的一块内存就被开辟使用，当函数返回又被释放。单线程只有一个栈，多线程系统则为每个线程都分配一个栈。

堆：可以动态地分配（系统）内存大小，生存期也不必事先告诉**编译器**，Java的垃圾收集器会自动收走这些不再使用的数据,c++、c中由程序员控制分配销毁。但缺点是，由于要在运行时动态分配内存，存取速度较慢。**堆式存储分配则专门负责在编译时或运行时模块入口处都无法确定存储要求的数据结构的内存分配**。



Java中语言自动管理栈和堆，程序员不能直接地设置栈或堆。

**在函数中定义的一些基本类型的变量（效率问题）和对象的引用变量都是在函数的栈内存中分配。**

当在一段代码块中定义一个变量时，java就在栈中为这个变量分配内存空间，当超过变量的作用域后，java会自动释放掉为该变量分配的内存空间，该内存空间可以立刻被另作他用。

堆内存用于存放由new创建的对象和数组。在堆中分配的内存，由**java虚拟机自动垃圾回收器**来管理。

1.申请后系统的响应

栈：只要栈的剩余空间大于所申请空间，系统将为程序提供内存，否则将报异常提示栈溢出。

堆：首先应该知道[操作系统](http://lib.csdn.net/base/operatingsystem)有一个记录空闲内存地址的链表，当系统收到程序的申请时，会遍历该链表，寻找第一个空间大于所申请空间的堆结点，然后将该结点从空闲结点链表中删除，并将该结点的空间分配给程序，另外，对于大多数系统，会在这块内存空间中的首地址处记录本次分配的大小，这样，代码中的delete语句才能正确的释放本内存空间。另外，由于找到的堆结点的大小不一定正好等于申请的大小，系统会自动的将多余的那部分重新放入空闲链表中。

2.申请效率的比较

栈：由系统自动分配，速度较快。但程序员是无法控制的。

堆：**是由new分配的内存**（程序里类似malloc），一般速度比较慢，而且容易产生内存碎片,不过用起来最方便。

3.申请大小的限制

栈：在Windows下，栈是向低地址扩展的数据结构，是一块连续的内存的区域。这句话的意思是栈顶的地址和栈的最大容量是系统预先规定好的

堆：堆是向高地址扩展的数据结构，是不连续的内存区域。这是由于系统是用链表来存储的空闲内存地址的，自然是不连续的，而链表的遍历方向是由低地址向高地址。堆的大小受限于计算机系统中有效的虚拟内存。由此可见，堆获得的空间比较灵活，也比较大。

堆内存用于存放由new创建的**对象和数组**。在堆中分配的内存，由java虚拟机自动垃圾回收器来管理。在堆中产生了一个数组或者对象后，还可以在栈中定义一个特殊的变量，这个变量的取值等于数组或者对象在堆内存中的首地址，在栈中的这个特殊的变量就变成了数组或者对象的引用变量，以后就可以在程序中**使用栈内存中的引用变量**来访问堆中的数组或者对象，引用变量相当于为数组或者对象起的一个别名，或者代号。

按照编译原理的观点,程序运行时的内存分配有三种策略,分别是静态的,栈式的,和堆式的.   
**静态存储分配**是指在**编译时**给他们分配固定的内存空间.

栈式存储分配也可称为动态存储分配,是由一个类似于堆栈的运行栈来实现的.和静态存储分配相反。

在栈式存储方案中,程序对数据区的需求**在编译时确定**,只有到运行的时候才分配,但是规定在运行中**进入一个程序模块时为其分配内存**.和我们在数据结构所熟知的栈一样,栈式存储分配按照先进后出的原则进行分配。

而**堆式存储分配则专门负责在编译时或运行时模块入口处都无法确定存储要求的数据结构的内存分配**,比如可变长度串和对象实例.堆由大片的可利用块或空闲块组成,堆中的内存可以按照任意顺序分配和释放.

内存的静态分配和动态分配的区别主要是两个

 一是时间不同。静态分配发生在程序编译和连接的时候。动态分配则发生在程序调入和执行的时候。

   二是空间不同。堆都是动态分配的，没有静态分配的堆。栈有2种分配方式：静态分配和动态分配。静态分配是编译器完成的，比如局部变量的分配。动态分配由函数malloc进行分配。不过栈的动态分配和堆不同，他的动态分配是由编译器进行释放，无需我们手工实现。

       对于一个进程的内存空间而言，可以在逻辑上分成3个部份：代码区，静态数据区和动态数据区。动态数据区一般就是“堆栈”。“栈(stack)”和“堆(heap)”是两种不同的动态数据区，栈是一种线性结构，堆是一种链式结构。进程的每个线程都有私有的“栈”，所以每个线程虽然代码一样，但本地变量的数据都是互不干扰。一个堆栈可以通过“基地址”和“栈顶”地址来描述。全局变量和静态变量分配在静态数据区，本地变量分配在动态数据区，即栈中。程序通过堆栈的基地址和偏移量来访问本地变量。

栈有一个很重要的特殊性，就是**存在栈中的数据可以共享（堆不可以）**。假设我们同时定义：   
　　int a = 3;   
　　int b = 3;   
　　编译器先处理int a = 3;首先它会在栈中创建一个变量为a的引用，然后查找栈中是否有3这个值，如果没找到，就将3存放进来，然后将a指向3。接着处理int b = 3;在创建完b的引用变量后，因为在栈中已经有3这个值，便将b直接指向3。这样，就出现了a与b同时均指向3的情况。这时，如果再令a=4;那么编译器会重新搜索栈中是否有4值，如果没有，则将4存放进来，并令a指向4;如果已经有了，则直接将a指向这个地址。因此a值的改变不会影响到b的值。要注意这种数据的共享与两个对象的引用同时指向一个对象的这种共享是不同的，因为这种情况a的修改并不会影响到b, 它是由编译器完成的，它有利于节省空间。而一个对象引用变量修改了这个对象的内部状态，会影响到另一个对象引用变量

|  |  |
| --- | --- |
| 例子：   |  | | --- | | String str1 = "abc";  String str2 = "abc";  System.out.println(str1==str2);//true |   注意， equals()是判断值是否相等。==号，根据JDK的说明，只有在两个引用都指向了同一个  对象时才返回真值。 |

String str1 = new String("abc");

String str2 = "abc";   
System.out.println(str1==str2); //false 创建了两个引用。创建了两个对象。

以上代码说明，只要是用new()来新建对象的，都会在堆中创建，而且其字符串是单独存值的，即使与栈中的数据相同，也不会与栈中的数据共享。