栈和堆详解

<http://blog.csdn.net/jasonwang18/article/details/70578647>

# 1、栈核对存放的内容

在函数中定义的一些基本类型的变量和对象的引用变量都在函数的栈内存中分配。 （函数的参数值，局部变量）

当在一段代码块定义一个变量时，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所分配的内存空间，该内存空间可以立即被另作他用。

堆内存用来存放由new创建的对象和数组。

     在堆中分配的内存，由Java[虚拟机](http://www.2cto.com/os/xuniji/)的自动垃圾回收器来管理。

# 2、栈和堆的区别和优势

栈的优势是，存取速度比堆要快，仅次于直接位于CPU中的寄存器。但缺点是，存在栈中的数据大小与生存期必须是确定的，缺乏灵活性。另外，栈数据可以共享，详见第3点。

堆的优势是可以动态地分配内存大小，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。但缺点是，由于要在运行时动态分配内存，存取速度较慢。

# 3. 栈的细讲

## 1、存放栈中的数据大小是固定的，生命周期也是固定的

一种是基本类型(primitive types), 共有8种，即int, short, long, byte, float, double, boolean, char(注意，并没有string的基本类型)。这种类型的定义是通过诸如int a = 3; long b = 255L;的形式来定义的，称为自动变量。值得注意的是，自动变量存的是字面值，不是类的实例，即不是类的引用，这里并没有类的存在。如int a = 3; 这里的a是一个指向int类型的引用，指向3这个字面值。这些字面值的数据，由于大小可知，生存期可知(这些字面值固定定义在某个程序块里面，程序块退出后，字段值就消失了)，出于追求速度的原因，就存在于栈中。

## 2、栈的数据共享

另外，栈有一个很重要的特殊性，就是存在栈中的数据可以共享。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，然后查找有没有字面值为3的地址，没找到，就开辟一个存放3这个字面值的地址，然后将a指向3的地址。

接着处理int b = 3；在创建完b的引用变量后，由于在栈中已经有3这个字面值，便将b直接指向3的地址。这样，就出现了a与b同时均指向3的情况。

特别注意的是，这种字面值的引用与类对象的引用不同。假定两个类对象的引用同时指向一个对象，如果一个对象引用变量修改了这个对象的内部状态，那么另一个对象引用变量也即刻反映出这个变化。相反，通过字面值的引用来修改其值，不会导致另一个指向此字面值的引用的值也跟着改变的情况。如上例，我们定义完a与 b的值后，再令a=4；那么，b不会等于4，还是等于3。在编译器内部，遇到a=4；时，它就会重新搜索栈中是否有4的字面值，如果没有，重新开辟地址存放4的值；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。

# 3、堆的简单讲解

另一种是包装类数据，如Integer, String, Double等将相应的基本数据类型包装起来的类。这些类数据全部存在于堆中，Java用new()语句来显式地告诉编译器，在运行时才根据需要动态创建，因此比较灵活，但缺点是要占用更多的时间。

# 4、String 变量的存储

String str1 = "abc"; String str2 = "abc";

System.out.println(str1==str2); //true

 注意，我们这里并不用str1.equals(str2)；的方式，因为这将比较两个字符串的值是否相等。==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。而我们在这里要看的是，str1与str2是否都指向了同一个对象。

结果说明，JVM创建了两个引用str1和str2，但只创建了一个对象，而且两个引用都指向了这个对象。其实同基本类型的变量

String、Integer、Long这些，都是final类型的类，任何一个线程都改变不了它们的值，

### 2、再修改原来代码：

String str1 = "abc";

String str2 = "abc";

str1 = "bcd";

String str3 = str1;

System.out.println(str3); //bcd

String str4 = "bcd";

System.out.println(str1 == str4); //true

str3 这个对象的引用直接指向str1所指向的对象(注意，str3并没有创建新对象)。当str1改完其值后，再创建一个String的引用str4，并指向因str1修改值而创建的新的对象。可以发现，这回str4也没有创建新的对象，从而再次实现栈中数据的共享。 我们再接着看以下的代码。

### 3、new String(“abc”)

String str1 = "abc";

String str2 = new String("abc");

 System.out.println(str1==str2); //false

创建了两个引用。创建了两个对象。两个引用分别指向不同的两个对象。 以上两段代码说明，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。

### 4、使用选择

使用String str = "abc"；的方式，可以在一定程度上提高程序的运行速度，因为JVM会自动根据栈中数据的实际情况来决定是否有必要创建新对象。而对于String str = new String("abc")；的代码，则一概在堆中创建新对象，而不管其字符串值是否相等，是否有必要创建新对象，从而加重了程序的负担。这个思想应该是享元模式的思想，但JDK的内部在这里实现是否应用了这个模式，不得而知。

(3)当比较包装类里面的数值是否相等时，用equals()方法；当测试两个包装类的引用是否指向同一个对象时，用==。