**[Java内存分配之堆、栈和常量池](http://www.cnblogs.com/SaraMoring/p/5687466.html)**

Java内存分配主要包括以下几个区域:

1. 寄存器：我们在程序中无法控制

2. 栈：存放基本类型的数据和对象的引用，但对象本身不存放在栈中，而是存放在堆中

3. 堆：存放用new产生的数据

4. 静态域：存放在对象中用static定义的静态成员

5. 常量池：存放常量

6. 非RAM(随机存取存储器)存储：硬盘等永久存储空间

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Java内存分配中的栈**

　　在函数中定义的一些基本类型的变量数据和对象的引用变量都在函数的栈内存中分配。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，当该变量退出该作用域后，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。

**Java内存分配中的堆**

　　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虚拟机的自动垃圾回收器来管理。

 　　在堆中产生了一个数组或对象后，还可以 在栈中定义一个特殊的变量，让栈中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。引用变量就相当于是为数组或对象起的一个名称，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）。这也是 Java 比较占内存的原因。

　　实际上，栈中的变量指向堆内存中的变量，这就是Java中的指针！

**常量池 (constant pool)**

　　常量池指的是在编译期被确定，并被保存在已编译的.class文件中的一些数据。除了包含代码中所定义的各种基本类型（如int、long等等）和对象型（如String及数组）的常量值(final)还包含一些以文本形式出现的符号引用，比如：

1. 类和接口的全限定名；
2. 字段的名称和描述符；
3. 方法和名称和描述符。

　　虚拟机必须为每个被装载的类型维护一个常量池。常量池就是该类型所用到常量的一个有序集和，包括直接常量（string,integer和 floating point常量）和对其他类型，字段和方法的符号引用。

　　对于String常量，它的值是在常量池中的。而JVM中的常量池在内存当中是以表的形式存在的， 对于String类型，有一张固定长度的CONSTANT\_String\_info表用来存储文字字符串值，注意：该表只存储文字字符串值，不存储符号引 用。说到这里，对常量池中的字符串值的存储位置应该有一个比较明了的理解了。

　　在程序执行的时候,常量池会储存在Method Area,而不是堆中。

**堆与栈**

　　Java的堆是一个运行时数据区,类的(对象从中分配空间。这些对象通过new、newarray、 anewarray和multianewarray等指令建立，它们不需要程序代码来显式的释放。堆是由垃圾回收来负责的，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，因为它是在运行时动态分配内存的，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。

　　栈的优势是，存取速度比堆要快，仅次于寄存器，栈数据可以共享。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。栈中主要存放一些基本类型的变量数据（int, short, long, byte, float, double, boolean, char）和对象句柄(引用)。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

　　这里我们主要关心栈，堆和常量池，对于栈和常量池中的对象可以共享，对于堆中的对象不可以共享。栈中的数据大小和生命周期是可以确定的，当没有引用指向数据时，这个数据就会消失。堆中的对象的由垃圾回收器负责回收，因此大小和生命周期不需要确定，具有很大的灵活性。

**字符串内存分配：**

　　对于字符串，其对象的引用都是存储在栈中的，如果是编译期已经创建好(直接用双引号定义的)的就存储在常量池中，如果是运行期（new出来的）才能确定的就存储在堆中。对于equals相等的字符串，在常量池中永远只有一份，在堆中有多份。

如以下代码：

[复制代码](javascript:void(0);)

String s1 = "china";

String s2 = "china";

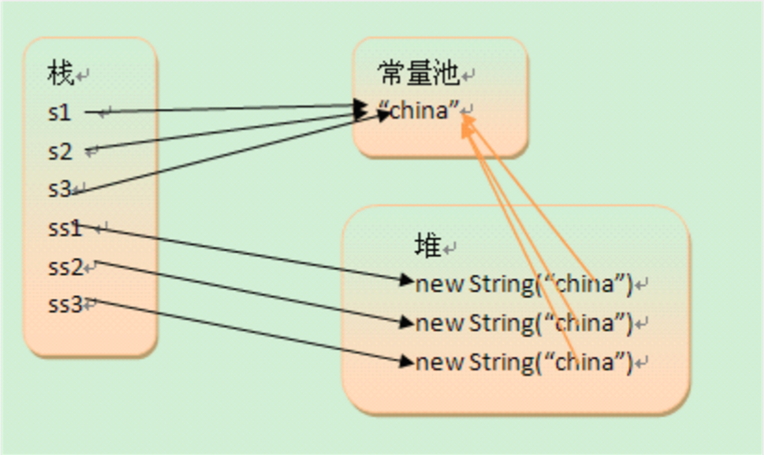
String s3 = "china";

String ss1 = new String("china");

String ss2 = new String("china");

String ss3 = new String("china");

[复制代码](javascript:void(0);)



　　这里解释一下黄色这3个箭头，对于通过new产生一个字符串（假设为“china”）时，会先去常量池中查找是否已经有了“china”对象，如果没有则在常量池中创建一个此字符串对象，然后堆中再创建一个常量池中此”china”对象的拷贝对象。

　　这也就是有道面试题：Strings=newString(“xyz”);产生几个对象？一个或两个，如果常量池中原来没有”xyz”,就是两个。

　　存在于.class文件中的常量池，在运行期被JVM装载，并且可以扩充。String的 intern()方法就是扩充常量池的 一个方法；当一个String实例str调用intern()方法时，Java 查找常量池中是否有相同Unicode的字符串常量，如果有，则返回其的引用，如果没有，则在常量池中增加一个Unicode等于str的字符串并返回它的引用

如下代码：

[复制代码](javascript:void(0);)

String s0= "kvill";

String s1=new String("kvill");

String s2=new String("kvill");

System.out.println( s0==s1 );

s1.intern();

s2=s2.intern(); //把常量池中"kvill"的引用赋给s2

System.out.println( s0==s1);

System.out.println( s0==s1.intern() );

System.out.println( s0==s2 );

[复制代码](javascript:void(0);)

输出结果：

false  
false  
true  
true

String常量池问题的几个例子：

[复制代码](javascript:void(0);)

【1】  
String a = "ab";

String bb = "b";

String b = "a" + bb;

System.out.println((a == b)); //result = false

【2】  
String a = "ab";

final String bb = "b";

String b = "a" + bb;

System.out.println((a == b)); //result = true

【3】  
String a = "ab";

final String bb = getBB();

String b = "a" + bb;

System.out.println((a == b)); //result = false

private static String getBB() {

return "b";

}

[复制代码](javascript:void(0);)

分析：

　　【1】中，JVM对于字符串引用，由于在字符串的"+"连接中，有字符串引用存在，而引用的值在程序编译期是无法确定的，即"a" + bb无法被编译器优化，只有在程序运行期来动态分配并将连接后的新地址赋给b。所以上面程序的结果也就为false。

　　【2】和【1】中唯一不同的是bb字符串加了final修饰，对于final修饰的变量，它在编译时被解析为常量值的一个本地拷贝存储到自己的常量池中或嵌入到它的字节码流中。所以此时的"a" + bb和"a" + "b"效果是一样的。故上面程序的结果为true。

　　【3】JVM对于字符串引用bb，它的值在编译期无法确定，只有在程序运行期调用方法后，将方法的返回值和"a"来动态连接并分配地址为b，故上面程序的结果为false。

结论：

　　字符串是一个特殊包装类,其引用是存放在栈里的,而对象内容必须根据创建方式不同定(常量池和堆).有的是编译期就已经创建好，存放在字符串常 量池中，而有的是运行时才被创建.使用new关键字，存放在堆中。

**基础类型的变量和常量在内存中的分配**

　　对于基础类型的变量和常量，变量和引用存储在栈中，常量存储在常量池中。

如以下代码：

[复制代码](javascript:void(0);)

int i1 = 9;

int i2 = 9;

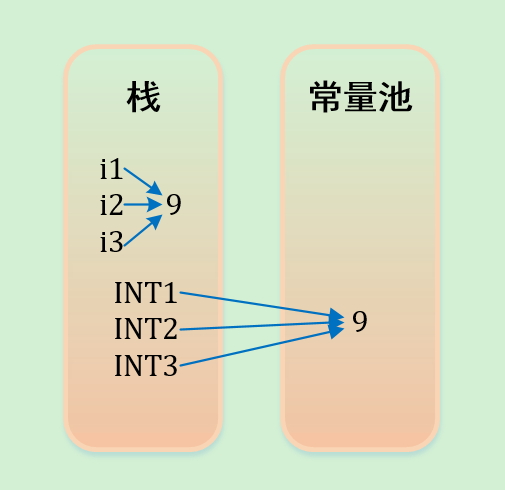
int i3 = 9;

final int INT1 = 9;

final int INT2 = 9;

final int INT3 = 9;

[复制代码](javascript:void(0);)



　　编译器先处理int i1 = 9；首先它会在栈中创建一个变量为i1的引用，然后查找栈中是否有9这个值，如果没找到，就将9存放进来，然后将i1指向9。接着处理int i2 = 9；在创建完i2的引用变量后，因为在栈中已经有9这个值，便将i2直接指向9。这样，就出现了i1与i2同时均指向9的情况。最后i3也指向这个9。

**成员变量和局部变量在内存中的分配**

　　对于成员变量和局部变量：成员变量就是方法外部，类的内部定义的变量；局部变量就是方法或语句块内部定义的变量。局部变量必须初始化。 形式参数是局部变量，局部变量的数据存在于栈内存中。栈内存中的局部变量随着方法的消失而消失。 成员变量存储在堆中的对象里面，由垃圾回收器负责回收。   如以下代码：

[复制代码](javascript:void(0);)

class BirthDate {

private int day;

private int month;

private int year;

public BirthDate(int d, int m, int y) {

day = d;

month = m;

year = y;

}

// 省略get,set方法………

}

public class Test {

public static void main(String args[]) {

int date = 9;

Test test = new Test();

test.change(date);

BirthDate d1 = new BirthDate(7, 7, 1970);

}

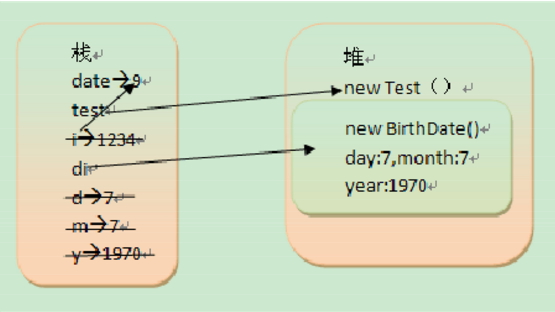
public void change(int i) {

i = 1234;

}

}

[复制代码](javascript:void(0);)



　　对于以上这段代码，date为局部变量，i,d,m,y都是形参为局部变量，day，month，year为成员变量。下面分析一下代码执行时候的变化：

1. main方法开始执行：int date = 9; date局部变量，基础类型，引用和值都存在栈中。
2. Test test = new Test();test为对象引用，存在栈中，对象(new Test())存在堆中。
3. test.change(date);  i为局部变量，引用和值存在栈中。当方法change执行完成后，i就会从栈中消失。
4. BirthDate d1= new BirthDate(7,7,1970); d1为对象引用，存在栈中，对象(new BirthDate())存在堆中，其中d，m，y为局部变量存储在栈中，且它们的类型为基础类型，因此它们的数据也存储在栈中。day,month,year为成员变量，它们存储在堆中(new BirthDate()里面)。当BirthDate构造方法执行完之后，d,m,y将从栈中消失。
5. main方法执行完之后，date变量，test，d1引用将从栈中消失，new Test(), new BirthDate()将等待垃圾回收。