# 概述

堆栈是两种数据结构，是一种数据项按序排列的数据结构，只能在一端进行插入和删除操作。堆为队列优先，先进先出（FIFO）。栈为先进后出（FILO）。

## 栈

**栈：**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。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所分配的内存空间，该内存空间可以立即被用作他用。

## 堆

**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者对象之后，还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）。

# 扩展

在堆中分配的内存，不是由程度员来进行释放的，而是由java虚拟机的垃圾回收机制来自己进行回收的。因为它是运行时动态分配内存的，所以生存期也不必告诉编译区，java的垃圾回收器会自动回收走不再使用的数据。但是，因为是动态分配内存，所以相对的也带来了缺点，那就是：存取速度较慢。

 当一段代码定义了一个变量时，java应在栈中为这个变量分配内存空间，当超过变量作用域时，java会自动释放掉为该变量分配的内存空间，该内存空间可台立即被另做它用。栈的优势是：存取速度快，仅次于CPU中的寄存器，同时，数据可以共享。缺点是：数据大小和生存期必须是确定的，缺乏灵活性。

 假设我们定义：

     a = 3;

     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字面值为3地址，如果没有找到，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，因为栈中已经有3这个字面值了，便把b直接指向3的地址。这样，就会出现a与b同时指向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这个字面值的地址，如果没有，则开辟一个存放3这个字面值的地址，然后将a指向4的地址；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b的值。

但是这种数据共享和两个对象的引用同时指向一个对象的引用是不相同的。如果两个对象的引用同时指向一个对象，如果一个对象的引用变量修改了这个对象的内部状态，那么另一个对象的引用也即刻反映出了这个对象的变化。相反，通过字面值的引用来修改其值，不会发生另一个指向该字面值的引用值也发生改变的情况。例如：上面我们令a和b同时等于3，然后令a等于4，则b还是等于3，不会发生变化。

String是一个特殊的包装类对象，可以通过String str =  new String("abc");的方式来创建，也可以用String str = "abc";来创建。第一种是通过new来创建的，所以会保存在堆中，每调用一次就会创建一个新的对象。当通过new产生一个字符串时，先去常量池中查看是否有"abc"对象，如果没有，则在常量池中创建一个"abc"对象，然后在堆中创建一个常量池中"abc"的copy对象。在这里曾经有一个面试题问：String str = new String("abc");产生了几个对象，那么答案是：如果常量池中已有"abc"对象，则产生了一个对象，就是堆中的对常量池"abc"的copy对象。如果常量池中没有"abc"对象，则还要在常量池中创建一个"abc"对象，所以就会产生两个对象。而第二种是先在栈中创建一个String对象的引用变量str的引用，然后找到栈中有没有存放值为"abc"的地址，如果没有，则开辟一个存放字面值为"abc"的地址，接着创建一个新的String类的对象o，并将o的字符串值指向这个地址，并且在栈中的这个地址旁边记录下这个引用的对象o，将str指向对象o的地址；如果已经有了值为"abc"的地址，将查找对象o,并返回o的地址。

比较对象里的值是否相等时，用equals,而测试两个包装类的引用是否指向同一个对象时，用==,例：

  String s1 = "abc";  
        String s2 = "abc";  
        System.out.println("s1==s2?"+(s1==s2));//true  
        System.out.println("s1.equals(s2)?"+s1.equals(s2));//true  
        String s3 = new String("abc");  
        String s4 = new String("abc");  
        System.out.println("s3==s4?"+(s3==s4));//false  
        System.out.println("s3.equals(s4)?"+s3.equals(s4));//true  
        System.out.println("s1==s4?"+(s1==s4));//false  
        System.out.println("s1.equals(s4)?"+s1.equals(s4));//true

当定义String s1 = "abc";时，编译器在栈中创建一个变量为s1的引用，然后开辟一个存放"abc"字面值的地址，将s1指向"abc"的地址，当处理String s2 = "abc"时，因为栈中已经有值为"abc"的引用，这时会将s2指向这个值为"abc"的引用，所以，当判断对象s1、s2的对象引用是否相同时，因为它们都引用值o "abc"的引用，所以返回true，同时，它们的值引用一样，所以判断值是否相同时，也返回true;

而当定义String s3 = new String("abc");String s4 = new String("abc");时，因为每new一个对象时，就会有堆中创建一个新的对象，所以当用==判断两个对象是否相等时，会返回false,因为两个对象的值是相等的，所以用equals判断值是，返回true。