JAVA基础：关于Java栈与堆的思考 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/145/2021_2022_JAVA_E5_9F_BA_E7_A1_80_c104_145241.htm 1.栈(stack)与堆(heap)都是Java用来在Ram中存放数据的地方。与C 不同，Java自动管理栈和堆，程序员不能直接地设置栈或堆。 2. 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. Java中的数据类型有两种。 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。 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。 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()语句来显示地告诉编译器，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是一个特殊的包装类数据。即可以用String str = new String("abc").的形式来创建，也可以用String str = "abc"；的形式来创建(作为对比，在JDK 5.0之前，你从未见过Integer i = 3.的表达式，因为类与字面值是不能通用的，除了String。而在JDK 5.0中，这种表达式是可以的！因为编译器在后台进行Integer i = new Integer(3)的转换)。前者是规范的类的创建过程，即在Java中，一切都是对象，而对象是类的实例，全部通过new()的形式来创建。Java 中的有些类，如DateFormat类，可以通过该类的getInstance()方法来返回一个新创建的类，似乎违反了此原则。其实不然。该类运用了单例模式来返回类的实例，只不过这个实例是在该类

内部通过new()来创建的，而getInstance()向外部隐藏了此细节。那为什么在String str = "abc"；中，并没有通过new()来创建实例，是不是违反了上述原则？其实没有。 5. 关于String str = "abc"的内部工作。Java内部将此语句转化为以下几个步骤：(1)先定义一个名为str的对String类的对象引用变量：String str；(2)在栈中查找有没有存放值为"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，而且在栈中这个地址旁边记下这个引用的对象o。如果已经有了值为"abc"的地址，则查找对象o，并返回o的地址。 (3)将str指向对象o的地址。 值得注意的是，一般String类中字符串值都是直接存值的。但像String str = "abc"；这种场合下，其字符串值却是保存了一个指向存在栈中数据的引用！ 为了更好地说明这个问题，我们可以通过以下的几个代码进行验证。 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。 我们再来更进一步，将以上代码改成： String str1 = "abc". String str2 = "abc". str1 = "bcd". System.out.println(str1 "," str2). //bcd, abc System.out.println(str1==str2). //false 这就是说，赋值的变化导致了类对象引用的变化，str1指向了另外一个新对象！而str2仍旧指向原来的对象。上例中，当我们将str1的值改为"bcd"时

，JVM发现在栈中没有存放该值的地址，便开辟了这个地址，并创建了一个新的对象，其字符串的值指向这个地址。 事实上，String类被设计成为不可改变(immutable)的类。如果你要改变其值，可以，但JVM在运行时根据新值悄悄创建了一个新对象，然后将这个对象的地址返回给原来类的引用。这个创建过程虽说是完全自动进行的，但它毕竟占用了更多的时间。在对时间要求比较敏感的环境中，会带有一定的不良影响。 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。 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。两个引用分别指向不同的两个对象。 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。 6. 数据类型包装类的值不可修改。不仅仅是String类的值不可修改，所有的数据类型包装类都不能更改其内部的值。 7. 结论与建议： (1)我们在使用诸如String str = "abc"；的格式定义类时，总是想当然地认为，我们创建

了String类的对象str。担心陷阱！对象可能并没有被创建！唯一可以肯定的是，指向 String类的引用被创建了。至于这个引用到底是否指向了一个新的对象，必须根据上下文来考虑，除非你通过new()方法来显要地创建一个新的对象。因此，更为准确的说法是，我们创建了一个指向String类的对象的引用变量str，这个对象引用变量指向了某个值为"abc"的String类。清醒地认识到这一点对排除程序中难以发现的bug是很有帮助的。 (2)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==。 (4)由于String类的immutable性质，当String变量需要经常变换其值时，应该考虑使用StringBuffer类，以提高程序效率。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com