Java中的String:水深几许计算机等级考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/645/2021_2022_Java_E4_B8_AD_E7_9A_84_c97_645417.htm 对Java中String的理解一直源于C 中的String，但越来越不对劲。 从字符串常量到String对象，到地址池的引出，再到Java中内存的分类，再到上一篇中讲到的Java编译器的优化,最后到《java语言规范》。 刚刚开始学Java,遇到这种问题，不知道是该庆幸呢，还是愈发感到忧烦。但个人还是倾向于前者，任何疑惑都有可能是一个起点，引向一片新的未知世界。从这个起点开始，可能会有更多的疑惑，但要解决这些疑惑，给自己一个基本满意的交代（最起码把疑惑给干掉），一直顺藤摸瓜下去，终究会有一个尽头，而这个过程，对于刚刚开始学的我来说将是一个很好的学习契机。 C 中的String： 字符串常量:本质上有一个const char* pstr就可以表征了； String类:自定义类,通过重载，=等运算符，实现其功能；一切很清晰明了。 Java中的String: （某本教材）所有用双引号括起的字符串常量（又称作字面常数）都被认为是对象。 在Java中引用类型变量间用==，判别2个引用类型变量是否是同一对象的引用。Java中没有运算符的重载(只有对于String的，从不同层面上看可以理解是运算符的重载与否)。 于是乎，关于String，==，，=的各种问题混乱不堪。 Process: 不清楚String str="abc".时看到了地址池的概念，于是乎寻找有关Java内存分配的内容。找到一篇还不错的文章： http://zy19880423.javaeye.com/blog/434179 Java内存分配: 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存储：硬盘等永久存储空间。 Tips: 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。(其实就是Java中的指针)。 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。（不像C 中那样，临时对象在程序运行到代码段外时会被立即析构掉，所以，返回临时对象的指针是危险的！）（Java则例外，可以返回临时对象，如果赋值给一个外部变量，则该对象仍然继续存在，不会被回收，只是其临时引用变量被干掉而已）。 常量池(constant pool)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。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。在程序执行的时候,常量池会储存在 Method Area,而不是堆中

。 String常量池可以再运行时填充，需要调用String.intern()；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。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；如果原先str引用的是堆中的对象：str=str.intern().//原先堆中的对象会变成垃圾。 