Java初学者都必须理解的六大问题 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/206/2021_2022_Java_E5_88_9D_E5_AD_A6_c104_206938.htm 对于这个系列里的问题，每个学Java的人都应该搞懂。当然，如果只是学Java玩玩就无所谓了。如果你认为自己已经超越初学者了，却不很懂这些问题，请将你自己重归初学者行列。 问题一：我声明了什么！ String s = "Hello world!". 许多人都做过这样的事情，但是，我们到底声明了什么？回答通常是：一个String，内容是“ Hello world! ”。这样模糊的回答通常是概念不清的根源。如果要准确的回答，一半的人大概会回答错误。 这个语句声明的是一个指向对象的引用，名为“ s ”，可以指向类型为String的任何对象，目前指向"Hello world!"这个String类型的对象。这就是真正发生的事情。我们并没有声明一个String对象，我们只是声明了一个只能指向String对象的引用变量。所以，如果在刚才那句语句后面，如果再运行一句： String string = s. 我们是声明了另外一个只能指向String对象的引用，名为string，并没有第二个对象产生，string还是指向原来那个对象，也就是，和s指向同一个对象。 问题二："=="和equals方法究竟有什么区别？ ==操作符专门用来比较变量的值是否相等。比较好理解的一点是： int a=10. int b=10. 则a==b将是true。 但不好理解的地方是： String a=new String("foo"). String b=new String("foo"). 则a==b将返回false。 根据前一帖说过，对象变量其实是一个引用，它们的值是指向对象所在的内存地址，而不是对象本身。a和b都使用了new操作符，意味着将在内存中产生两个内容为"foo"的字符串，既然是“ 两个 ”，它们自然

位于不同的内存地址。a和b的值其实是两个不同的内存地址的值，所以使用"=="操作符，结果会是false。诚然，a和b所指的对象，它们的内容都是"foo"，应该是“ 相等 ”，但是==操作符并不涉及到对象内容的比较。 对象内容的比较，正是equals方法做的事。 看一下Object对象的equals方法是如何实现的： boolean equals(Object o){ return this==o. } Object对象默认使用了==操作符。所以如果你自创的类没有覆盖equals方法，那你的类使用equals和使用==会得到同样的结果。同样也可以看出，Object的equals方法没有达到equals方法应该达到的目标：比较两个对象内容是否相等。因为答案应该由类的创建者决定，所以Object把这个任务留给了类的创建者。 看一下一个极端的类： Class Monster{ private String content. ... boolean equals(Object another){ return true.} } 我覆盖了equals方法。这个实现会导致无论Monster实例内容如何，它们之间的比较永远返回true。 所以当你是用equals方法判断对象的内容是否相等，请不要想当然。因为可能你认为相等，而这个类的作者不这样认为，而类的equals方法的实现是由他掌握的。如果你需要使用equals方法，或者使用任何基于散列码的集合（HashSet,HashMap,HashTable），请察看一下java doc以确认这个类的equals逻辑是如何实现的。 问题三：String到底变了没有？ 没有。因为String被设计成不可变(immutable)类，所以它的所有对象都是不可变对象。请看下列代码： String s = "Hello". s = s " world!". s所指向的对象是否改变了呢？从本系列第一篇的结论很容易导出这个结论。我们来看看发生了什么事情。在这段代码中，s原先指向一个String对象，内容是"Hello"，然后我们对s进行了 操作，那么s所指向的那个对

象是否发生了改变呢？答案是没有。这时，s不指向原来那个对象了，而指向了另一个String对象，内容为"Hello world!"，原来那个对象还存在于内存之中，只是s这个引用变量不再指向它了。 通过上面的说明，我们很容易导出另一个结论，如果经常对字符串进行各种各样的修改，或者说，不可预见的修改，那么使用String来代表字符串的话会引起很大的内存开销。因为String对象建立之后不能再改变，所以对于每一个不同的字符串，都需要一个String对象来表示。这时，应该考虑使用StringBuffer类，它允许修改，而不是每个不同的字符串都要生成一个新的对象。并且，这两种类的对象转换十分容易。 同时，我们还可以知道，如果要使用内容相同的字符串，不必每次都new一个String。例如我们要在构造器中对一个名叫s的String引用变量进行初始化，把它设置为初始值，应当这样做： public class Demo { private String s. ... public Demo { s = "Initial Value". } ... } 而非 s = new String("Initial Value"). 后者每次都会调用构造器，生成新对象，性能低下且内存开销大，并且没有意义，因为String对象不可改变，所以对于内容相同的字符串，只要一个String对象来表示就可以了。也就说，多次调用上面的构造器创建多个对象，他们的String类型属性s都指向同一个对象。 上面的结论还基于这样一个事实：对于字符串常量，如果内容相同，Java认为它们代表同一个String对象。而用关键字new调用构造器，总是会创建一个新的对象，无论内容是否相同。 至于为什么要把String类设计成不可变类，是它的用途决定的。其实不只String，很多Java标准类库中的类都是不可变的。在开发一个系统的时候，我们有时候也需要设计不可变类，来传递一组相关的值，这也是面向对象思

想的体现。不可变类有一些优点，比如因为它的对象是只读的，所以多线程并发访问也不会有任何问题。当然也有一些缺点，比如每个不同的状态都要一个对象来代表，可能会造成性能上的问题。所以Java标准类库还提供了一个可变版本，即StringBuffer。