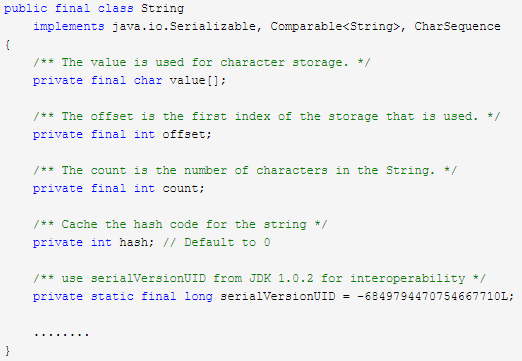
# String

## 一

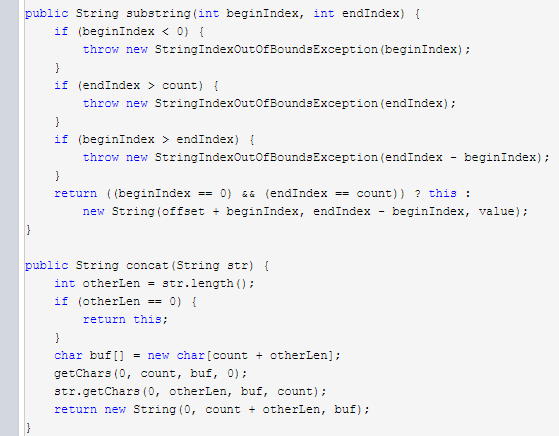
String类的部分源码



从图可以看出几点：

1. String类是final类，也即意味着String类不能被继承，并且它的成员方法都默认为final方法。在Java中，被final修饰的类是不允许被继承的，并且该类中的成员都默认为final方法。
2. 上面列举了String的成员属性，从上面可以看出String类其实是通过char数组来保存字符串的。

String一些方法实现代码



从上面方法看出，无论是sub，concat还是replace操作都不是在原有的字符串上进行的，而是重新生成了一个新的字符串对象。也就是说进行这些操作后，最原始的字符串并没有被改变。

String对象一旦创建就是固定不变的了。对String对象的任何改变都不影响到原对象，相关的任何change操作都会生成新的对象。

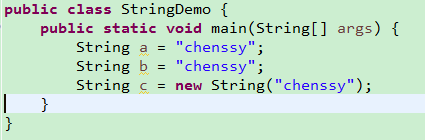
## 二 字符串常量池

字符串的分配和其他对象分配一样，需要消耗高昂的时间和空间，而且字符串我们使用的非常多。JVM为了提高性能和减少内存的开销，在实例化字符串时进行了一些优化：使用字符串常量池。每当我们创建字符串时，JVM会首先检查字符串常量池，如果该字符串已经存在常量池中，那么直接返回常量池中的实例引用。如果字符串不存在常量池中，就会实例化该字符串并且将其放到常量池中。由于String字符串的不可变性，可以知道常量池中一定不存在两个相同的字符串。

Java中的常量池，实际上分为两种形态：静态常量池和运行时常量池。

所谓静态常量池，即\*.class文件中的常量池，class文件中的常量池不仅仅包含字符串（数字）字面量，还包含类，方法的信息，占用class文件绝大部分空间。而运行时常量，则是jvm虚拟机在完成类装载操作后，将class文件中的常量池载入到内存中，并保存在方法区中，我们常说的常量池，就是指方法区中的运行时常量池。

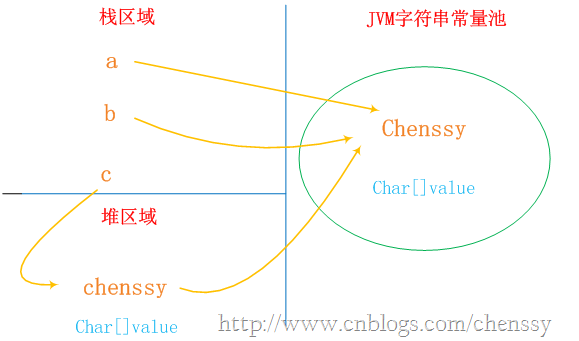
看下面程序：



1. b和字面上的chenssy都是指向JVM字符串常量池中的”chenssy”对象，他们指向同一个对象。

new关键字一定会产生一个对象chenssy（注意这个chenssy和上面

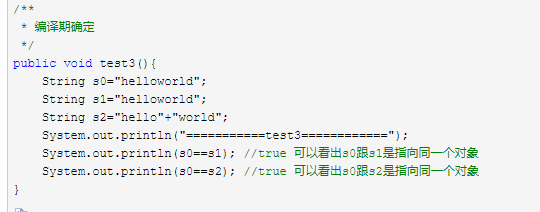
的chenssy不同）,同时这个对象是存储在堆中。所以上面应该产生了两个对 象：保存在栈中的c和保存堆中chenssy。但是在Java中根本不存在两个完 全一模一样的字符串对象。故堆中的chenssy应该是引用字符串常量池中 chenssy。所以c,chenssy,池chenssy的关系应该是：c->chenssy->池chenssy。 关系如下：



总结：虽然a,b,c,chenssy是不同的对象，但是从String的内部结构我们是可 以理解上面的，String c = new String(“chenssy”);虽然c的内容是创建在堆中， 但是他的内部value还是指向JVM常量池的chenssy的value，它构造chenssy 时所用的参数依然是chenssy字符串常量。

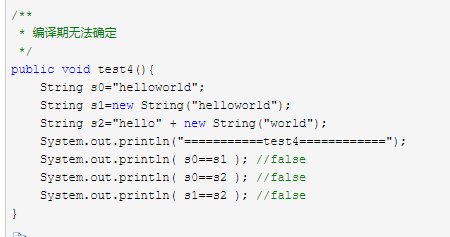
看几个例子：

1)

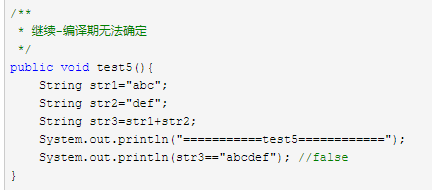


分析：因为s0和s1中的”helloworld”都是字符串常量，他们在编译期就被确定了，所以s0==s1为true;而”hello”和”world”也都是字符串常量，当一个字符串常量由多个字符串常量连接而成时，它自己肯定也是常量，所以s2同样在编译期被解析为一个字符串常量，所以s2也是常量池中”helloworld”的一个引用，所以s0==s1==s2。

2）



3）



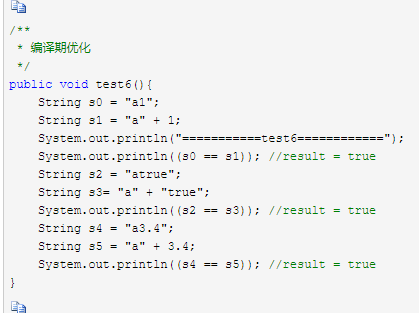
执行上述代码，结果为:false

分析：因为str3指向堆中的”abcdef”对象，而”abcdef”是字符串池中的对象，所以结果为false。JVM对String str = “abc”对象放在”abc”对象放在常量池中是在编译时做的，而String str3 = str1 + str2 是在运行时刻才能知道的。new对象也是在运行时才做的。而这段代码总共创建了5个对象，字符串池中两个，堆中三个。+运算符会在堆中建立两个String对象，这两个对象的值分别为”abc”，”def”,也就是说从字符串池中复制这两个值，然后在堆中创建两个对象，然后再建立对象str3然后将”abcdef”的堆地址赋值给str3。

步骤:

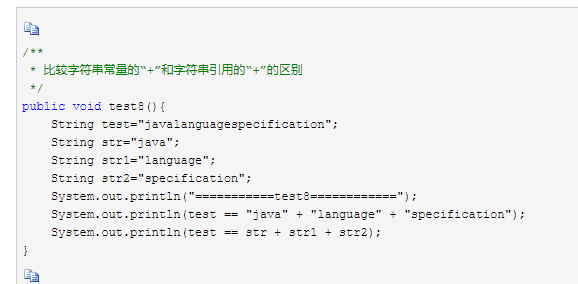
1. 栈中开辟一块中间存放引用str1，str1指向池中String常量”abc”。
2. 栈中开辟一块中间存放引用str2，str2指向池中String常量”def”。
3. 栈中开辟一块中间存放引用str3。
4. str1 + str2通过StringBuilder的最后一步toString()方法还原一个新的String对象”abcdef”，因此堆中开辟一块空间存放此对象。
5. 引用str3指向堆中(str1 + str2)所还原的新String对象。
6. Str3指向的对象在堆中，而常量”abcdef”在池中，输出位false。

4)



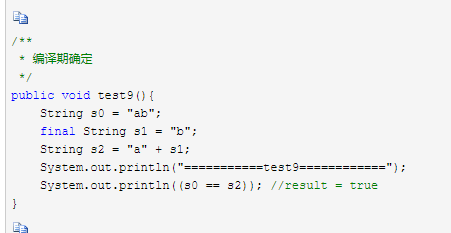
分析：在程序编译期，JVM就将常量字符串的”+”连接优化为连接后的值，拿”a” + 1来说，经编译器优化后在class中就已经是a1。在编译期其字符串常量的值就确定下来了。

5)



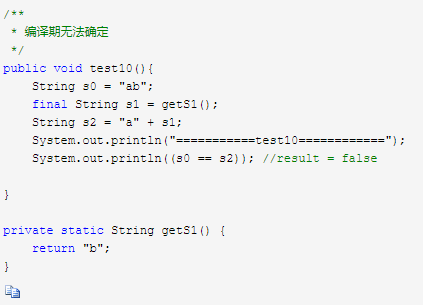
结果为true, false

6)



分析：对于final修饰的变量，它在编译时被解析为一个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”+s1和”a”+”b”效果是一样的。故上面程序的结果为true。

7)



结果为：false

分析：虽然s1被final修饰了，但是由于其赋值是通过方法调用返回的，那么它的值只能在运行期间确定，因此s0和s2指向的不是同一个对象，故上面程序结果为false。

## 三 总结

### String类初始化后是不可变的 String使用private final char value[] 来实现字符串的存储， 也就是说String对象创建之后，就不能修改此对象中存储的字符串内容， 就是因为如此，才说String类型是不可变的。

1. 引用变量与对象

A aa;

这个语句声明一个类A的引用变量aa（我们常常称之为句柄），而对象一般通过new创建。所以aa仅仅是一个引用变量,它不是对象。

1. 创建字符串的方式
2. 使用””引号创建字符串。
3. 使用new关键字创建字符串。

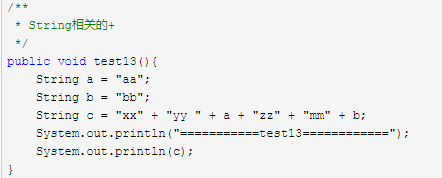
结合上面例子，总结如下:

1. 单独使用””引号创建的字符串都是常量，编译期就已经确定存储 到String Pool中；
2. 使用new String(“”)创建的对象会存储到heap中，是运行期创建 的。new创建字符串时首先查看池中是否有相同值的字符串，如果 有，则拷贝一份到堆中，然后返回堆中的地址；如果池中没有，则 在堆中创建一份，然后返回堆中的地址（注意，此时不需要从堆中 复制到池中，否则，将使得堆中的字符串永远是池中的子集，导致 浪费池的空间）
3. 使用只包含常量的字符串连接符如”aa”+”aa”创建的也是常量，编 译期就能确定，已经确定存储到String Pool中；
4. 使用包含变量的字符串连接符如”aa”+s1创建的对象是运行期才 创建的，存储在heap中
5. 使用String不一定创建对象

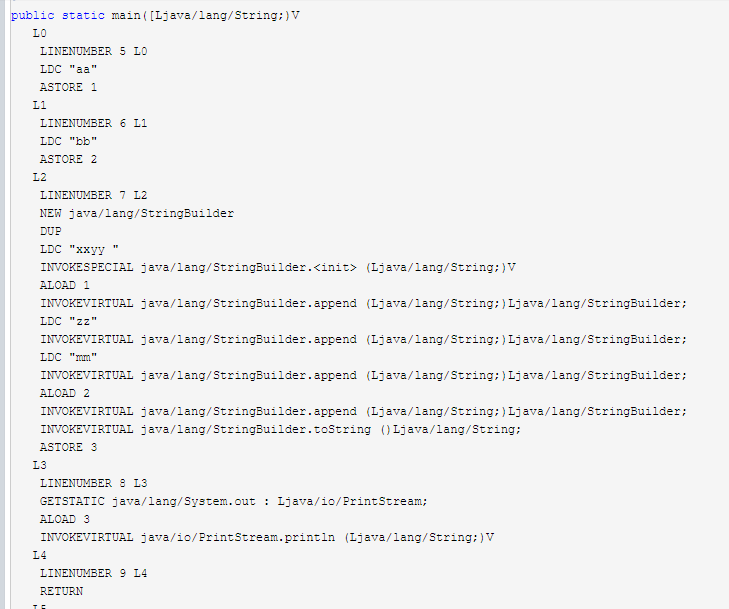
在执行到双引号包含字符串的语句时，如String a = “123”,JVM会先到常量池里查找，如果有的话返回常量池里的这个实例的引用，否则的话创建一个新实例并置入常量池里。只有通过new（）方法才能保证每次都创建一个新的对象。New String(“123”)创建了两个对象！！

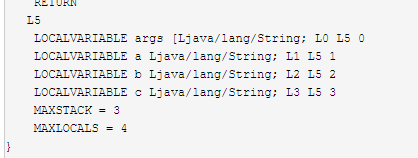
1. 关于equals 和 ==
2. 对于==,如果作用于基本数据类型的变量，则直接比较其存储的”值” 是否相等;如果作用于引用类型的变量(String)，则比较的是所指向的对象 的地址(即是否指向同一个对象)。
3. Equals方法是基类Object中的方法，因此对于所有的继承于Object 的类都会有该方法。在Object类中，rquals方法就是用来比较两个对象 的引用是否相等，即是否指向同一个对象。
4. 对于equals方法，注意：equals方法不能作用于基本数据类型的变量。 如果没有对equals方法进行重写，则比较的是引用类型的变量所指向的 对象的地址，而String类对equals方法进行了重写，用来比较指向的字 符串对象所存储的字符串是否相等。其他的一些类诸如Double， Date,Integer等，都对equals方法进行了重写用来比较指向的对象所存储 的内容是否相等。
5. String相关的+:

String中的+常用于字符串的连接。



编译运行后字节码部分如下





显然，通过字节码我们可以得出如下几点结论:

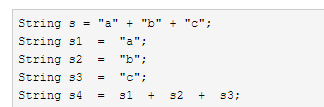
1. String中使用+字符串连接符进行字符串连接时，连接操作最开始时如 果都是字符串常量，编译后将尽可能多的直接将字符串常亮连接起来， 形成新的字符串常量参与后续连接(通过反编译工具也可以方便的看出)
2. 接下来字符串连接是从左到右依次进行，对于不同的字符串，首先以 最左边的字符串为参数创建StringBuilder对象，然后依次对右边进行 append操作，最后将StringBuilder对象通过toString()方法也就是说

String c = “xx”+”yy”+a+”zz”+”mm”+b;实质上实现过程是：

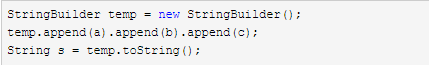
String c

=new StringBuilder(“xxyy”).append(a).append(“zz”).append(“mm”).append(b). toString()

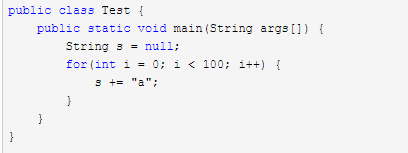
1. String的不可变性导致字符串变量使用+号的代价：



分析：变量s的创建等价于String s = “abc”;由上面例子可知编译器进 行了优化，这里只创建了一个对象。由上面的例子也可以知道s4不能在 编译期优化，其对象创建相当于：

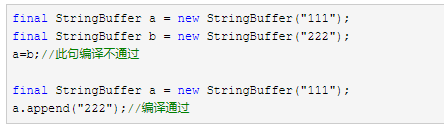


由上面的分析结果，不难推断出String采用连接运算符(+)效率低下原因分析，形如下面代码



每做一次 + 就产生个StringBuilder对象，然后append后就扔掉。下次循环再到达时重新产生个StringBuilder对象，然后append字符串，如此循环直至结束。如果我们直接采用StringBuilder对象进行append的话，可以节省N-1次创建和销毁对象的时间。所以对于在循环中要进行字符串连接的应用，一般都是StringBuffer或StringBuilder对象来执行append操作。

1. String,StringBuffer,StringBuilder的区别
2. 可变与不可变：String是不可变字符串对象，StringBuilder和StringBuffer是可变字符串对象（其内部的字符串数组长度可变）。
3. 是否多线程安全：String中的对象是不可变的，也就可以理解为常量，显然线程安全。StringBuffer与StringBuilder中的方法和功能完全是等价的，只是StringBuffer中的方法大都采用了synchronized关键字进行修饰，因此是线程安全的，而StringBuilder没有这个修饰，可以被认为是非线程安全的。
4. String中的final用法和理解



可见，final只对引用的”值”（即内存地址）有效，它迫使引用只能指向初始指向的那个对象，改变它的指向会导致编译期错误。至于他所指向的对象的变化，final不负责。

1. 关于String str = new String(“abc”)创建了多少个对象？

2个对象说法比较片面。运行期间只创建了一个对象。即new的对象，在类加载过程中，字符串常量池也创建了一个。所以整个编译+运行期创建了两个对象。

1. 字符串池的优缺点：

字符串池的有点就是避免了相同内容的创建，节省了内存，省去了创建相同字符串的时间，同时提升了性能。另一方面，字符串池的缺点就是牺牲了JVM在常量池中遍历对象所需要的时间，不过其时间成本相对而言比较低。