# 回顾

1. lang包:java内置的基础包,程序会帮我们自动导入
2. enum:安全检查功能,规范参数值
3. 包装类:8个

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Byte | Short | Integer | Long | Float | Double | Boolean | Character |
| byte | short | int | long | float | double | boolean | char |

4.为什么要使用包装类,它是为了替代基本数据类型吗

1.和面向对象的思想一致,基本数据类型没有方法属性

2.和其他类型进行转换,需要用到包装类中的方法

3.不是替代,只是有些场合基本数据类型使用不了,比如集合....

5.自动拆箱: 自动将 包装类 变成 基本数据类型

自动装箱: 自动将 基本数据类型 变成 包装类

Jdk1.5以上才支持自动拆装箱: 拆: Integer i=10; int a=i; 装:int a=1,Integer i=a;

6.Integer:将字符串转整数:int a=Integer.parseInt(string) ; Integer i=Integer.valueof(string)

7.Character:判断是字母,数字,大写,小写

8.String: **[charAt](mk:@MSITStore:E:\\标准化教案\\API帮助文档\\JDK_API_1_6.CHM::/java/lang/String.html" \l "charAt(int))(int index) [contains](mk:@MSITStore:E:\\标准化教案\\API帮助文档\\JDK_API_1_6.CHM::/java/lang/String.html" \l "contains(java.lang.CharSequence))() [equals](mk:@MSITStore:E:\\标准化教案\\API帮助文档\\JDK_API_1_6.CHM::/java/lang/String.html" \l "equals(java.lang.Object))() [length](mk:@MSITStore:E:\\标准化教案\\API帮助文档\\JDK_API_1_6.CHM::/java/lang/String.html" \l "length())() [isEmpty](mk:@MSITStore:E:\\标准化教案\\API帮助文档\\JDK_API_1_6.CHM::/java/lang/String.html" \l "isEmpty())() [valueOf](mk:@MSITStore:E:\\标准化教案\\API帮助文档\\JDK_API_1_6.CHM::/java/lang/String.html" \l "valueOf(int))(int i) [equalsIgnoreCase](mk:@MSITStore:E:\\API帮助文档\\jdk1.6\\JDK_API_1_6.CHM::/java/lang/../../java/lang/String.html" \l "equalsIgnoreCase(java.lang.String))**([String](mk:@MSITStore:E:\\API帮助文档\\jdk1.6\\JDK_API_1_6.CHM::/java/lang/../../java/lang/String.html" \o "java.lang 中的类) anotherString) [toUpperCase](mk:@MSITStore:E:\\API帮助文档\\jdk1.6\\JDK_API_1_6.CHM::/java/lang/../../java/lang/String.html" \l "toUpperCase())() [toLowerCase](mk:@MSITStore:E:\\API帮助文档\\jdk1.6\\JDK_API_1_6.CHM::/java/lang/../../java/lang/String.html" \l "toLowerCase())()

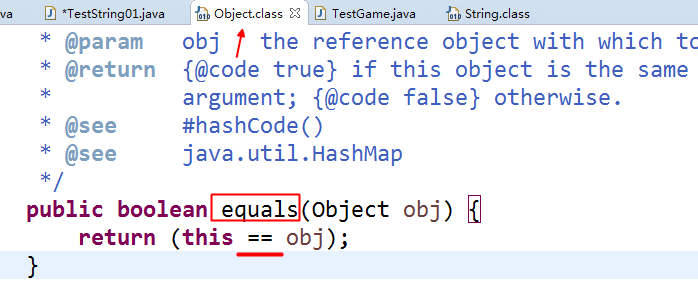
# String

## 【面试题】String的==和equals()判断方法

1. == 判断的是两个字符串的地址是否相同

2. equals(ObjectanObject)判断的是两个字符串的内容是否相同

注:1：Object类的equals()方法，其实判断是地址，而不是内容。



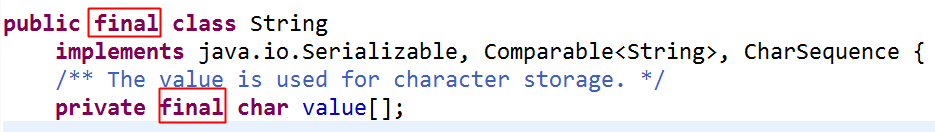
String继承Object类，重写了equals方法，变成了判断的是字符串的内容。

注意2：关于equals()方法的代码规范：尽量使用非空对象来调用equals()方法，避免出现空指针异常

## 【面试题】string的不可变性:

1. String在JDK源码中，该类是被final修饰，不可以被继承。

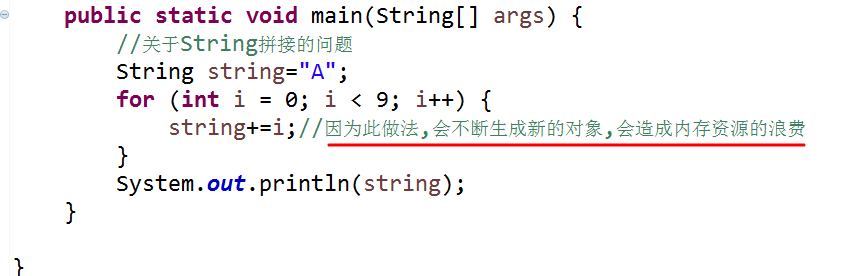
String底层是字符数组，被final修饰，表示不可变。



1. 用代码演示不变性



## 【面试题】String拼接问题:



不能使用String进行字符串的拼接动作

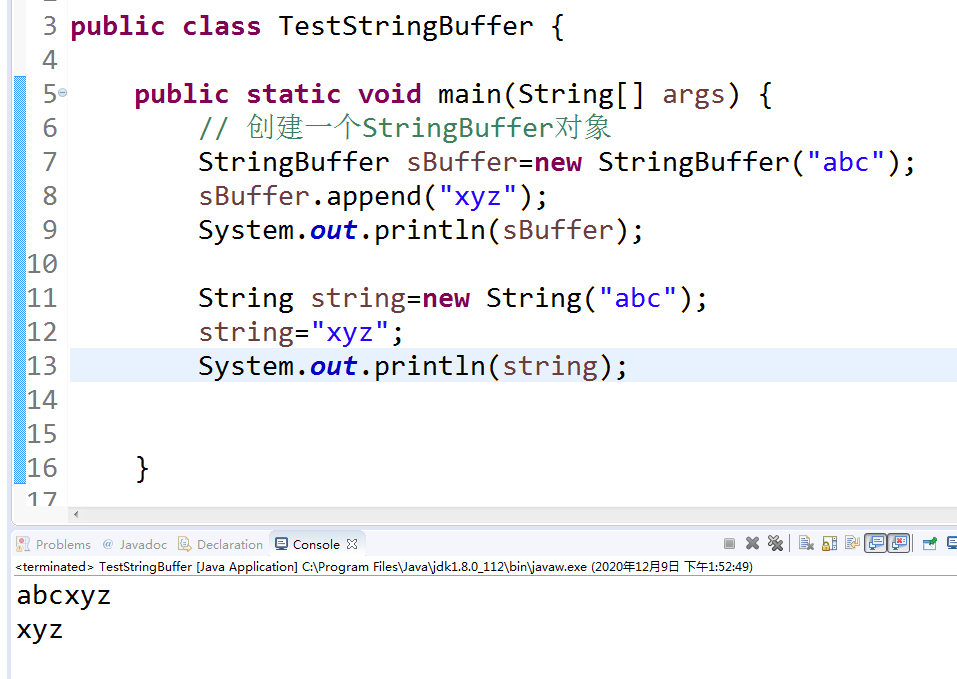
# 类 StringBuffer

概念:线程安全的可变字符序列。一个类似于 [String](mk:@MSITStore:E:\\API帮助文档\\jdk1.6\\JDK_API_1_6.CHM::/java/lang/../../java/lang/String.html" \o "java.lang 中的类) 的字符串缓冲区，但能修改。虽然在任意时间点上它都包含某种特定的字符序列，但通过某些方法调用可以改变该序列的长度和内容。

|  |  |
| --- | --- |
| **构造方法摘要** | |
| **[StringBuffer](mk:@MSITStore:E:\\API帮助文档\\jdk1.6\\JDK_API_1_6.CHM::/java/lang/../../java/lang/StringBuffer.html" \l "StringBuffer())**()            构造一个其中不带字符的字符串缓冲区，其初始容量为 16 个字符。 |  |
| **[StringBuffer](mk:@MSITStore:E:\\API帮助文档\\jdk1.6\\JDK_API_1_6.CHM::/java/lang/../../java/lang/StringBuffer.html" \l "StringBuffer(java.lang.String))**([String](mk:@MSITStore:E:\\API帮助文档\\jdk1.6\\JDK_API_1_6.CHM::/java/lang/../../java/lang/String.html" \o "java.lang 中的类) str)            构造一个字符串缓冲区，并将其内容初始化为指定的字符串内容。 |  |

常见方法:

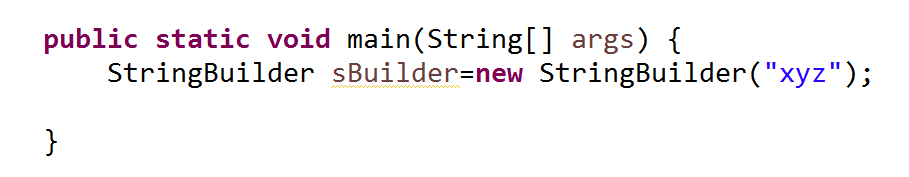
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| int | **[capacity](mk:@MSITStore:E:\\API帮助文档\\jdk1.6\\JDK_API_1_6.CHM::/java/lang/../../java/lang/StringBuffer.html" \l "capacity())**()            返回当前容量。 | | | |
| [StringBuffer](mk:@MSITStore:E:\\标准化教案\\帮助文档\\API帮助文档\\jdk1.6\\JDK_API_1_6.CHM::/java/lang/../../java/lang/StringBuffer.html" \o "java.lang 中的类) | | **[append](mk:@MSITStore:E:\\标准化教案\\帮助文档\\API帮助文档\\jdk1.6\\JDK_API_1_6.CHM::/java/lang/../../java/lang/StringBuffer.html" \l "append(java.lang.String))**([String](mk:@MSITStore:E:\\标准化教案\\帮助文档\\API帮助文档\\jdk1.6\\JDK_API_1_6.CHM::/java/lang/../../java/lang/String.html" \o "java.lang 中的类) str)  将指定的字符串追加到此字符序列。 | | |
| char | | | **[charAt](mk:@MSITStore:E:\\API帮助文档\\jdk1.6\\JDK_API_1_6.CHM::/java/lang/../../java/lang/StringBuffer.html" \l "charAt(int))**(int index)            返回此序列中指定索引处的 char 值。 |
| [StringBuffer](mk:@MSITStore:E:\\API帮助文档\\jdk1.6\\JDK_API_1_6.CHM::/java/lang/../../java/lang/StringBuffer.html" \o "java.lang 中的类) | | **[reverse](mk:@MSITStore:E:\\API帮助文档\\jdk1.6\\JDK_API_1_6.CHM::/java/lang/../../java/lang/StringBuffer.html" \l "reverse())**()            将此字符序列用其反转形式取代。 | | |



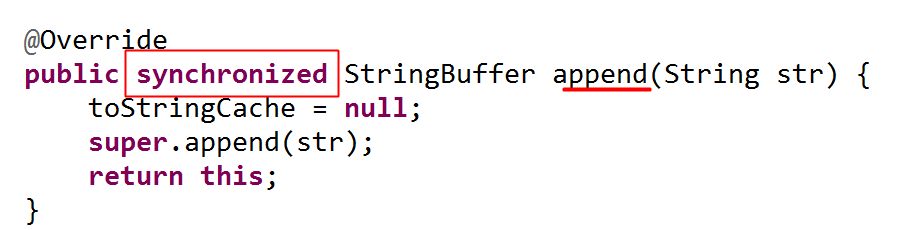
# 类 StringBuilder

一个可变的字符序列。这个类提供了一个API兼容 StringBuffer，但无法保证同步.非线程安全

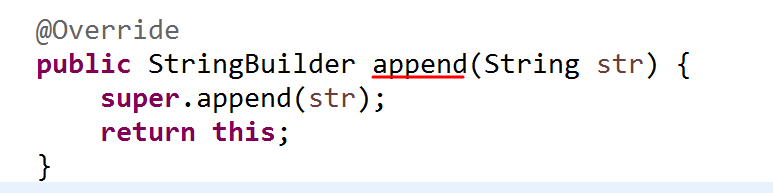
|  |
| --- |
| **构造方法** |
| **Constructor and Description** |
| [StringBuilder](mk:@MSITStore:E:\\API帮助文档\\jdk1.8中文\\jdk1.8中文百度.CHM::/java/lang/../../java/lang/StringBuilder.html" \l "StringBuilder--)()  构造一个没有字符的字符串生成器，并构造了16个字符的初始容量。 |
| [StringBuilder](mk:@MSITStore:E:\\API帮助文档\\jdk1.8中文\\jdk1.8中文百度.CHM::/java/lang/../../java/lang/StringBuilder.html" \l "StringBuilder-java.lang.String-)([String](mk:@MSITStore:E:\\API帮助文档\\jdk1.8中文\\jdk1.8中文百度.CHM::/java/lang/../../java/lang/String.html" \o "class in java.lang) str)  构造一个初始化为指定字符串的内容的字符串生成器。 |



Stringbuffer:有一个线程同步的关键字,效率比较低,相对安全



StringBuilder:无线程同步的关键字,效率高



StringBuﬀer与StringBuilder共同特征：可变性，不同：一个线程安全，一个非线程安全

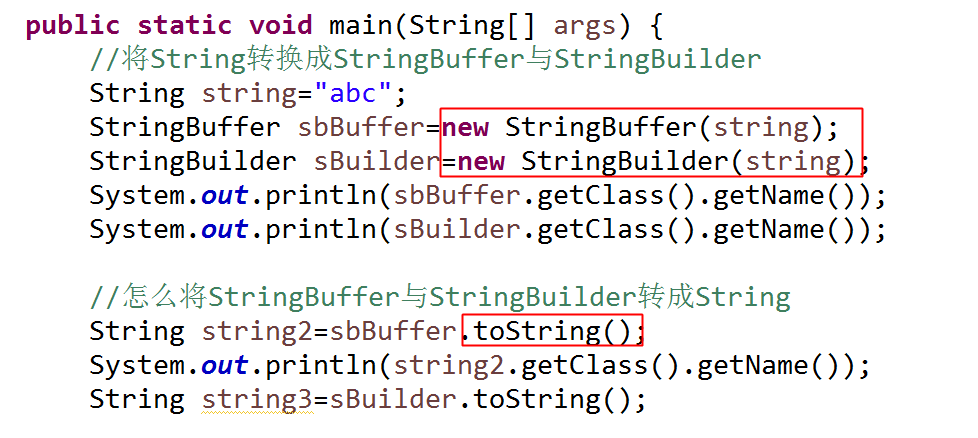
## 【面试题：String和StringBuffer和StringBuilder区别】

String StringBuffer StringBuilder

是否可变 不可变 可变 可变

线程同步安全 不同步 同步 不同步

## 问题: StringBuffer/ StringBuilder和String如何转换



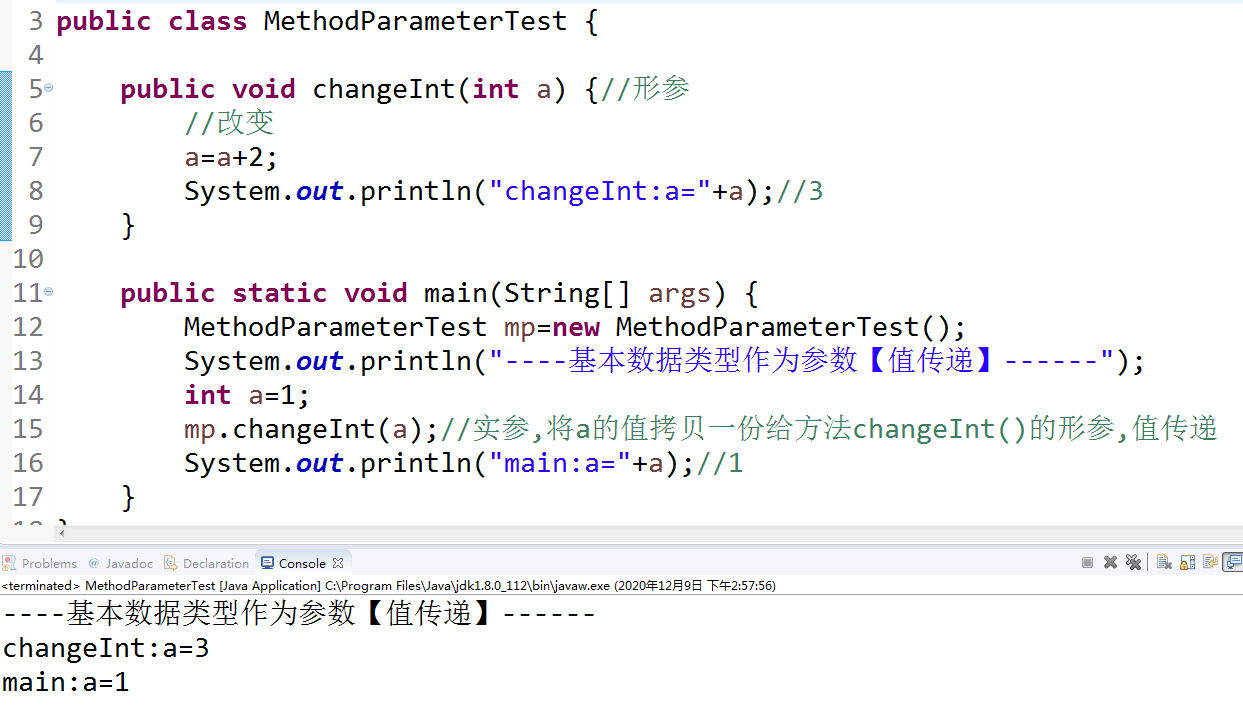
## 问题: StringBuffer和StringBuilder相比为什么支持线程同步

Stringbuffer:有一个线程同步的关键字,效率比较低,相对安全

StringBuilder:无线程同步的关键字,效率高

# 方法的参数传递

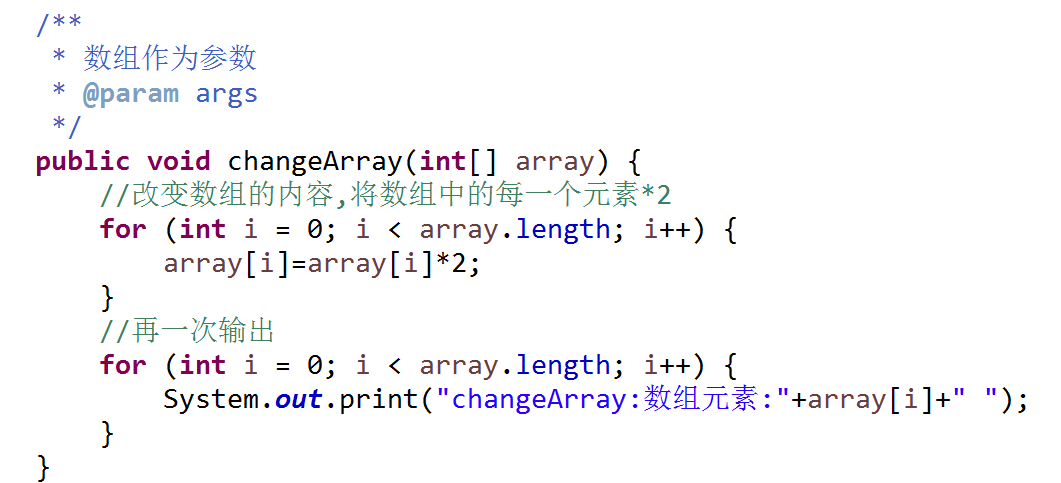
### 基本数据类型作为参数【值传递】：



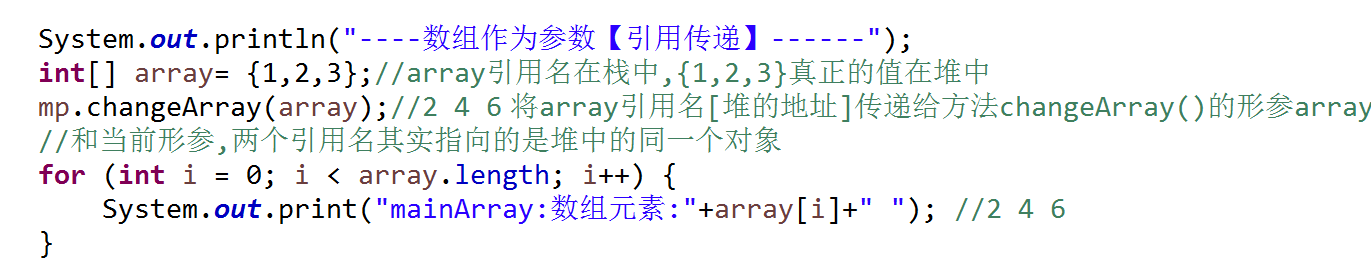
**基本数据类型作为参数,传递的是值的一个副本给形参,在方法内对形参的操作，不影响实参的值。**

### 引用数据类型(数组、类)作为参数【引用传递】

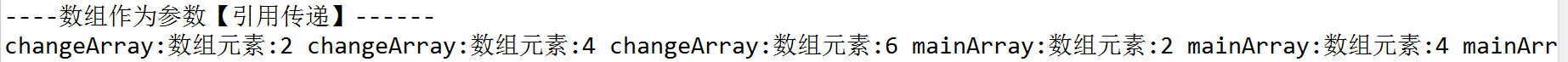
将数组作为参数：



测试类：



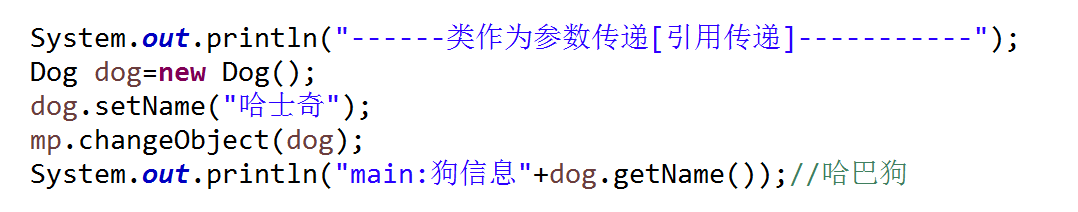
测试结果：



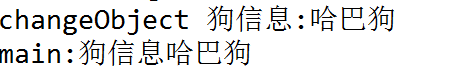
把类作为参数：



测试类：



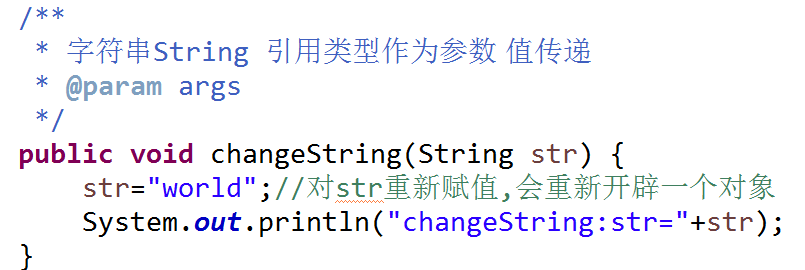
测试结果：



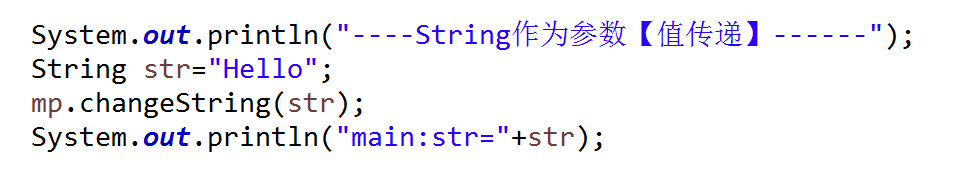
**数组、类等引用数据类型作为参数，传递的引用名的值其实是堆的地址给形参，则实参和形参两个引用名指向堆中的同一个对象,在方法体中对该地址中的值改变就会影响实参。**

### Sting字符串作为参数【值传递】

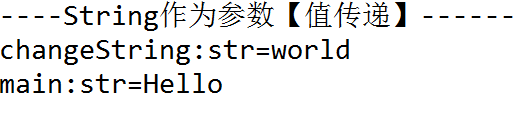
函数定义和函数调用：



测试类：



测试结果：



**String传递的时候，其实也是将实参引用名中的堆的地址传递给形参，因为String具有不可变性,当方法体内对该地址的字符串重新赋值,则相当于在内存在重新创建了一个对象,不影响实参。**

## 总结：值传递与引用传递

值传递：

(形式参数类型是基本数据类型)：

方法调用时，实际参数把它的值传递给对应的形式参数，形式参数只是用实际参数的值初始化自己的存储单元内容，是两个不同的存储单元，所以方法执行中形式参数值的改变不影响实际参数的值。

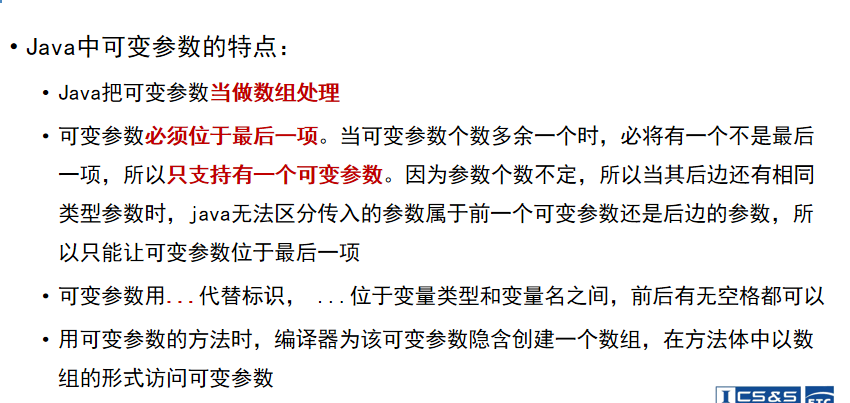
引用传递

(形式参数类型是引用数据类型参数)：

也称为传地址。方法调用时，实际参数是对象(或数组)，这时实际参数与形式参数指向同一个地址，在方法执行中，对形式参数的操作实际上就是对实际参数的操作，这个结果在方法结束后被保留了下来，所以方法执行中形式参数的改变将会影响实际参数。

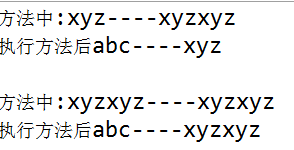
## 可变参数

Java1.5增加了可变参数特性，适用于参数个数不确定，类型确定的情况

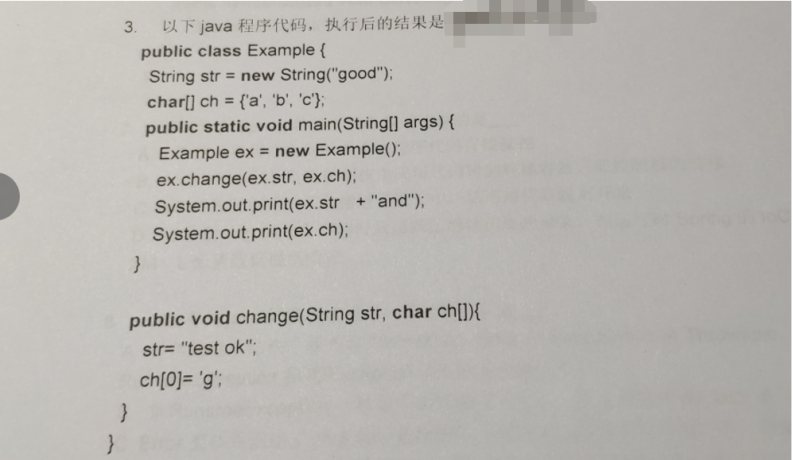


## 练习：



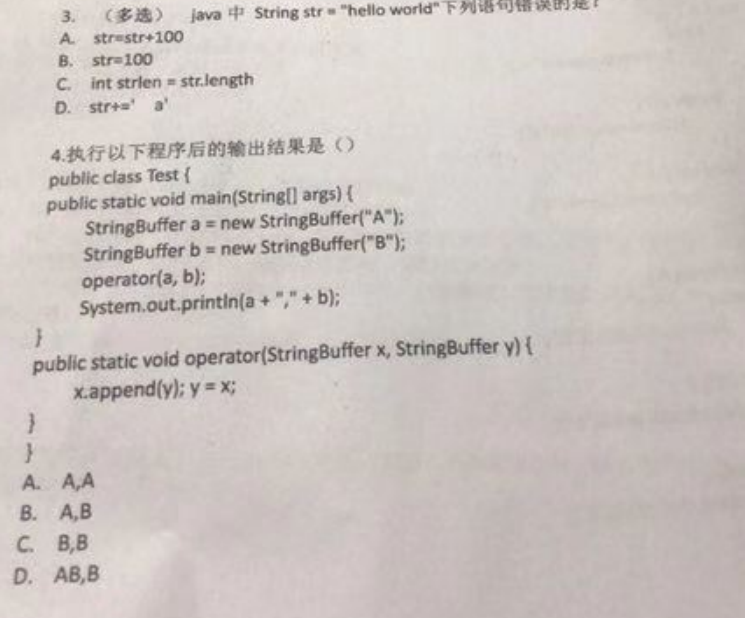


## 【笔试题】



考察： String值传递 字符数组引用传递

答案：goodandgbc



1. BCD
2. d

# 字符串类小结:

1) String和StringBuffer、StringBuilder相比，String是不可变的，String的每次修改操作都是在内存中重新new一个对象出来，而StringBuffer、StringBuilder则不用，并且提供了一定的缓存功能，默认16个字节数组的大小，超过默认的数组长度时，则扩容为原来字节数组的长度\*2+2。

1. . StringBuffer和StringBuilder相比，StringBuffer是synchronized的，是线程安全的，而StringBuilder是非线程安全的，单线程情况下性能更好一点；使用StringBuffer和StringBuilder时，可以适当考虑下初始化大小，减少扩容的次数，提高代码的高效性。
2. 方法传递:值传递(将基本数据类型,String作为参数)->形参与实参的值互不影响,

引用传递(将引用类型作为参数)->形参与实参的值相互影响