Java 基础（5）：String 性质深入解析

对于初学者来说，很容易误认为 String 对象是可以改变的，特别是+链接时，对象似乎真的改变了。然而，String 对象一经创建就不可以修改。接下来，我们一步步**分析 String 是怎么维护其不可改变的性质；**

1.手段一：final 类 和 final 的私有成员

我们先看一下 String 的部分源码：

```
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence {
    /** The value is used for character storage. */
    private final char value[];
    /** Cache the hash code for the string */
    private int hash; // Default to 0
    /** use serialVersionUID from JDK 1.0.2 for interoperability */
    private static final long serialVersionUID = -6849794470754667710L;
}
```

我们可以发现 **String 是一个 final 类，且 3 个成员都是私有的**，这就意味着 String 是不能被继承的，这就防止出现：程序员通过继承重写 String 类的方法的手段来使得 String 类是“可变的”的情况。

从源码发现，每个 String 对象维护着一个 char 数组 —— 私有成员 value。**数组 value 是 String 的底层数组，用于存储字符串的内容，而且是 private final** ，但是数组是引用类型，所以只能限制引用不改变而已，也就是说数组元素的值是可以改变的，而且 String 有一个可以传入数组的构造方法，那么我们可不可以通过修改外部 char 数组元素的方式来“修改”String 的内容呢？

我们来做一个实验，如下：

```
public static void main(String[] args) {
        char[] arr = new char[]{'a','b','c','d'};
        String str = new String(arr);
        arr[3]='e';
        System.out.println("str= "+str);
        System.out.println("arr[]= "+Arrays.toString(arr));
}
```

运行结果

```
str= abcd
arr[]= [a, b, c, e]
```

结果与我们所想不一样。字符串 str 使用数组 arr 来构造一个对象，当数组 arr 修改其元素值后，字符串 str 并没有跟着改变。那就看一下这个构造方法是怎么处理的：

```
public String(char value[]) {
    this.value = Arrays.copyOf(value, value.length);
}
```

原来 **String 在使用外部 char 数组构造对象时，是重新复制了一份外部 char 数组**，从而不会让外部 char 数组的改变影响到 String 对象。

2.手段二：改变即创建对象的方法

从上面的分析我们知道 ,我们是无法从外部修改 String 对象的 ,那么可不可能使用 String 提供的方法 ,因为有不少方法看起来是可以改变 String 对象的 ,如 replace()、replaceAll()、substring()等。我们以 substring()为例，看一下源码：

```
public String substring(int beginIndex, int endIndex) {
    //........
    return ((beginIndex == 0) && (endIndex == value.length)) ? this : new
   String(value, beginIndex, subLen);
}
```

从源码可以看出，如果不是切割整个字符串的话，就会新建一个对象。也就是说，**只要与原字符串不相等，就会新建一个 String 对象。**

扩展

基本类型的包装类跟 String 很相似的，都是 final 类，都是不可改变的对象，以及维护着一个存储内容的 private final 成员。如 Integer 类：

```
public final class Integer extends Number implements Comparable<Integer> {
     private final int value;
}
```

我们先来看一个例子：

```
public class MyTest {
    public static void main(String[] args) {
        String s = "Love You";
        String s2 = "Love"+" You";
        String s3 = s2 + "";
        String s4 = new String("Love You");
```

```
        System.out.println("s == s2 "+(s==s2));
        System.out.println("s == s3 "+(s==s3));
        System.out.println("s == s4 "+(s==s4));
    }
}
```

运行结果：

```
s == s2  true
s == s3  false
s == s4  false
```

是不是对运行结果感觉很不解。别急，我们来慢慢理清楚。首先，我们要知道编译器有个优点：在编译期间会尽可能地优化代码，所以**能由编译器完成的计算，就不会等到运行时计算，如常量表达式的计算就是在编译期间完成的。**所以，s2 的结果其实在编译期间就已经计算出来了，与 s 的值是一样，所以两者相等，即都属于字面常量，在类加载时创建并维护在字符串常量池中。但 s3 的表达式中含有变量 s2,只能是运行时才能执行计算,也就是说，在运行时才计算结果，在堆中创建对象，自然与 s 不相等。而 s4 使用 new 直接在堆中创建对象，更不可能相等。

那在运行期间，是如何完成 String 的+号链接操作的呢，要知道 String 对象可是不可改变的对象。我们使用 jad 命令 jad MyTest.class 反编译上面例子的 calss 文件回 java 代码，来看看究竟是怎么实现的：

```
public class MyTest{
    public MyTest(){ }
    public static void main(String args[]){
        String s = "Love You";
        String s2 = "Love You";//已经得到计算结果
        String s3 = (new StringBuilder(String.valueOf(s2))).toString();
        String s4 = new String("Love You");
        System.out.println((new StringBuilder("s == s2 ")).append(s ==
s2).toString());
        System.out.println((new StringBuilder("s == s3 ")).append(s ==
s3).toString());
        System.out.println((new StringBuilder("s == s4 ")).append(s ==
s4).toString());
    }
}
```

可以看出，编译器将 + 号处理成了 StringBuilder.append()方法。也就是说，在运行期间，链接字符串的计算都是通过 创建 StringBuilder 对象，调用 append()方法来完成的，而且是每一个链接字符串的表达式都要创建一个 StringBuilder 对象。因此**对于循环中反复执行字符串链接时，应该考虑直接使用 StringBuilder 来代替 + 链接，避免重复创建 StringBuilder 的性能开销。**

字符串常量池

常量池可以参考我上一篇文章，此处不会深入，只讲解与 String 相关的部分。
字符串常量池的内容大部分来源于编译得到的字符串字面常量。在运行期间同样也会增加.

- **String intern()：**

  返回字符串对象的规范化表示形式。

  一个初始为空的字符串池，它由类 String 私有地维护。

  当调用 intern 方法时，如果池已经包含一个等于此 String 对象的字符串（用 equals(Object) 方法确定），则返回池中的字符串。否则，将此 String 对象添加到池中，并返回此 String 对象的引用。

  它遵循以下规则：对于任意两个字符串 s 和 t，当且仅当 s.equals(t) 为 true 时，s.intern() == t.intern() 才为 true。

  另外一点值得注意的是，虽然 String.intern()的返回值永远等于字符串常量。但这并不代表在系统的每时每刻，相同的字符串的 intern()返回都会是一样的（虽然在 95%以上的情况下，都是相同的）。因为存在这么一种可能：在一次 intern()调用之后，该字符串在某一个时刻被回收，之后，再进行一次 intern()调用，那么字面量相同的字符串重新被加入常量池，但是引用位置已经不同。

  String 也是遵守 equals 的标准的，也就是 s.equals(s1) 为 true，则 s.hashCode()==s1.hashCode() 也为 true。此处并不关注 eqauls 方法，而是讲解 hashCode()方法，String.hashCode()有点意思，而且在面试中也可能被问到。先来看一下代码：

```
public int hashCode() {
    int h = hash;
    if (h == 0 && value.length > 0) {
        char val[] = value;
        for (int i = 0; i < value.length; i++) {
            h = 31 * h + val[i];
        }
        hash = h;
    }
    return h;
}
```

为什么要选 31 作为乘数呢？

从网上的资料来看，一般有如下两个原因：

- 31 是一个不大不小的质数，是作为 hashCode 乘子的优选质数之一。另外一些相近的质数，比如 37、41、43 等等，也都是不错的选择。那么为啥偏偏选中了 31 呢？请看第二个原因。

- 31 可以被 JVM 优化，31 * i = (i << 5) - i。