

57从零开始学Java57之String字符串的底层原理

前言

配套开源项目资料

一. Spring源码中的final关键词

1. final的特点

2. String源码解读

2.1 final修饰的String类

2.2 final修饰的value[]属性

二. String的不可变性

1. 实验案例

2. 结果剖析

三. String真的不可变吗？

1. 实验案例

2. 结果剖析

四. 结语

1. 不可变性的优点

2. 不可变性的缺点

五. 今日作业

作者：孙玉昌，昵称【**一一哥**】，另外【**壹壹哥**】也是我哦

千锋教育高级教研员、CSDN博客专家、万粉博主、阿里云专家博主、掘金优质作者

前言

在之前的两篇文章中，**壹哥**给大家介绍了String字符串及其常用的API方法、常用编码、正则表达式等内容，但这些内容都是停留在”如何用“的阶段，没有涉及到”为什么“的层面。实际上，我们在求职时，面试官很喜欢问我们关于String的一些原理性知识，比如String的不可变性、字符串的内存分配等。为了让大家更好地应对面试，并理解String的底层设计，接下来**壹哥**会给大家聊聊String的一些原理，比如String为什么具有不可变性？

全文大约【4000】字，不说废话，只讲可以让你学到技术、明白原理的纯干货！本文带有丰富的案例及配图视频，让你更好地理解 and 运用文中的技术概念，并可以给你带来具有足够启迪的思考.....

配套开源项目资料

Github:

d/LearnJava

GitHub – SunLtd/LearnJava

Contribute to SunLtd/LearnJava development by creating an account on GitHub.

GitHub

Gitee:



一一哥/从零开始学Java

从零开始学Java系列 稀土掘金专栏地址: <https://juejin.cn/column/7175082165548351546> CSDN专...

Gitee

一. Spring源码中的final关键词

为了弄清楚String为什么具有不可变性，我们先来看看String的源码，尤其是源码中带有final关键词的地方。

1. final的特点

为了更好地理解String相关的内容，在阅读String源码之前，我们先来复习一下final关键词有哪些特点，因为在String中会涉及到很多final相关的内容。

1. final关键词修饰的类不可以被其他类继承，但是该类本身可以继承其他类，通俗的说就是这个类可以有父类，但是不能有子类；
2. final关键词修饰的方法不可以被覆盖重写，但是可以被继承使用；
3. final关键词修饰的基本数据类型变量称为常量，只能被赋值一次；

4. final关键词修饰的引用数据类型的变量值为地址值，地址值不能改变，但是地址内的数据对象可以被改变；
5. final关键词修饰的成员变量，需要在创建对象前赋值，否则会报错(即需要在定义时直接赋值，如果是在构造方法中赋值，则多个构造方法均需赋值)。

复习了final的特点之后，接下来我们就可以阅读String的源码了。

2. String源码解读

接下来就请大家跟着壹哥来看看String源码中关于不可变性的内容吧。

2.1 final修饰的String类

```
Java | 复制代码
1  /**
2   * .....其他略.....
3   *
4   * Strings are constant; their values cannot be changed after they
5   * are created. String buffers support mutable strings.
6   * Because String objects are immutable they can be shared. For example:
7   *
8   * .....其他略.....
9   *
10  */
11  public final class String
12      implements java.io.Serializable, Comparable<String>, CharSequence {
13
14      .....
```

壹哥先对上面的源码及其注释进行简单的解释：

- final：请参考第1小节对final特点的介绍；
- Serializable：用于序列化；
- Comparable<String>：默认的比较器；
- CharSequence：提供对字符序列进行统一、只读的操作。

从这一段源码及注释中，我们可以得出如下结论：

- String类用final关键字修饰，说明String不可被继承；
- String字符串是常量，字符串的值一旦被创建，就不能被改变；

- String字符串缓冲区支持可变字符串；
- 因为String对象是不可变的，所以它们是可以被共享的。

2.2 final修饰的value[]属性

```
1 public final class String
2     implements java.io.Serializable, Comparable<String>, CharSequence {
3     /** The value is used for character storage. */
4     private final char value[];
5
6     .....
```

从源码中可以看出，value[]是一个私有的字符数组，String类其实就是通过这个char数组来保存字符串内容的。简单的说，**我们定义的字符串会被拆成一个一个的字符，这些字符都被存放在这个value字符数组里面。**

这里的value[]数组被final修饰，初始化之后就不能再被更改。但是大家注意，**我们这里说的value[]不可变，指的是value的引用地址不可变，但是value数组里面的数据元素其实是可变的！**这是因为value是数组类型，根据我们之前学过的知识，**value的引用地址会分配在栈中，而其对应的数据是在常量池中保存的。所以我们说String不可变，指的就是value在栈中的引用地址不可变，而不是说常量池中数组本身的数据元素不可变。**

另外我们要注意，Java中的字符串常量池，用来存储字符串字面量！但是由于JDK版本的不同，**常量池的位置也不同：**

JDK 6 及以下版本的字符串常量池是在方法区(Perm Gen)中，此时常量池中存储的是字符串对象；在 JDK 8.0 中，方法区(永久代被元空间取代了)；

JDK 7、8以后的字符串常量池被转移到了堆中，此时常量池存储的就是字符串对象的引用，而不是字符串对象本身。

至此，壹哥就带各位把String类中的核心源码分析完了，接下来我们进一步分析String不可变的原因，及其他底层原理设计。

二. String的不可变性

1. 实验案例

了解了上面的这些核心源码之后，接下来 壹哥 再带各位来验证一下，看看String到底能不能变！我先给各位来一段案例代码，代码案例如下图所示。

```
//String不可变
String s = "yiyige";
System.out.println("s = " + s);

s = "yyg";
System.out.println("s = " + s);
}
```

StringTest04 > main()

StringTest04 ×

C:\Program Files\Java\jdk1.8.0_152\bin\

s = yiyige

s = yyg

千锋教育-孙玉昌

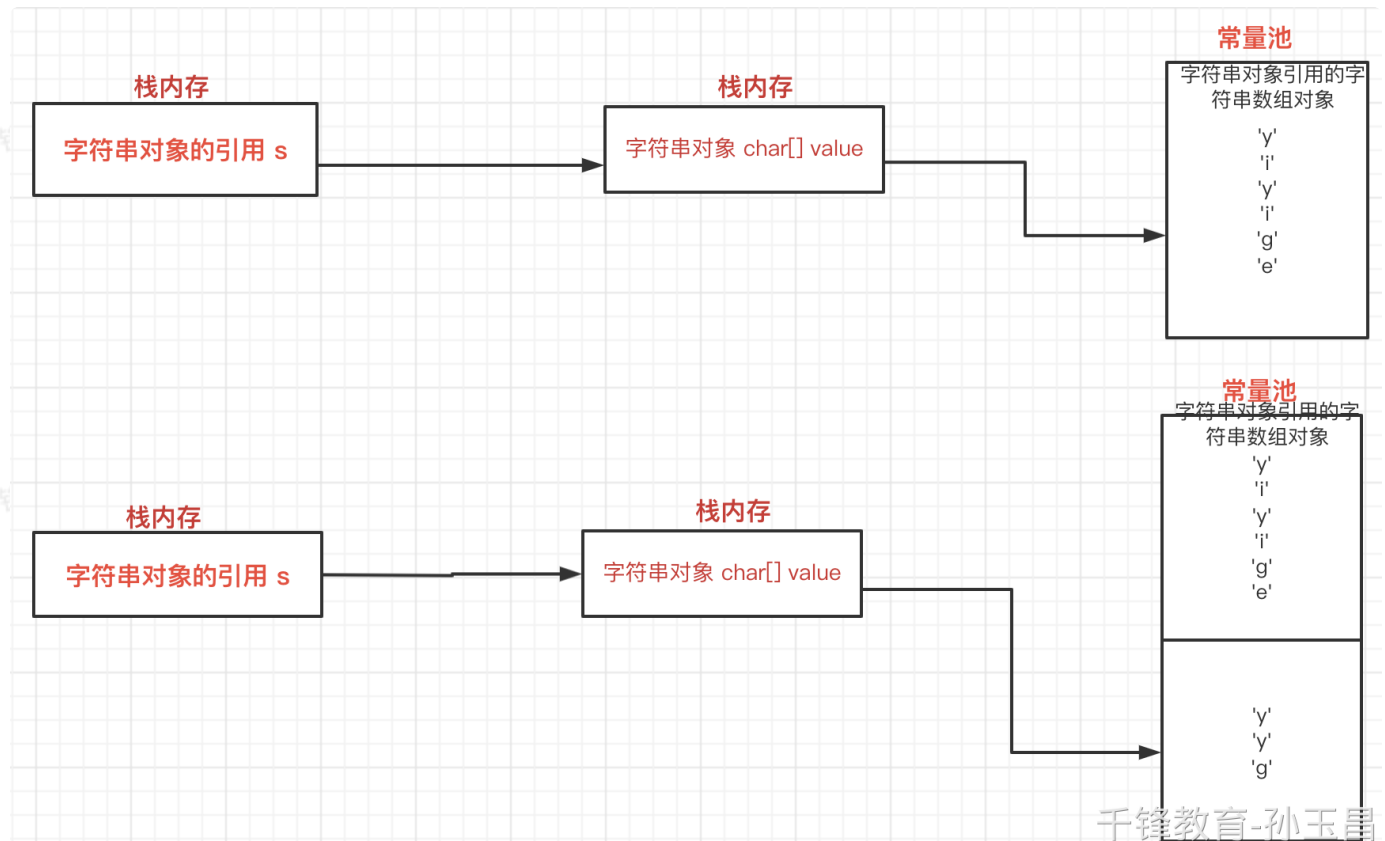
结果s的内容变了，好像是啪啪打脸了？？？！！！咋回事，壹哥 不是说了String不可变吗？怎么这么快就翻车打脸了？别急，让我们好好来分析一下。

2. 结果剖析

首先我们从结果上来看String s 变量的结果好像改变了，但为什么我们又说String是不可变的呢？

要想明白这个问题，我们得先弄清楚一个点，即**引用和值的区别**！在上面的代码中，我们先是创建了一个 "yiyige" 为内容的字符串引用s，s其实先是指向了value对象，而value对象则指向存储了 "y, i, y, i, g, e" 字符的字符数组。因为value被final修饰，所以value的值不可被更改。因此，

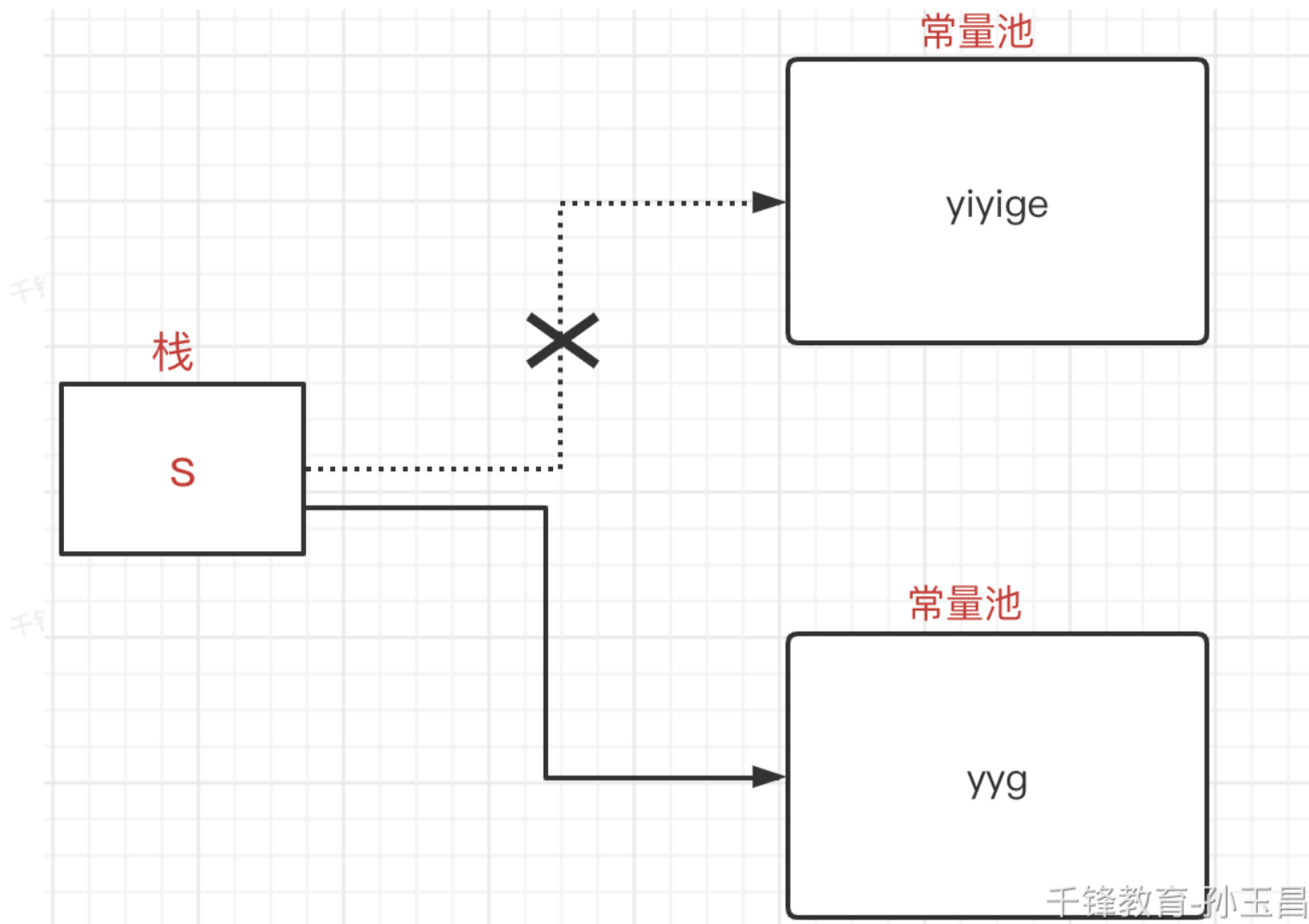
上面代码中改变的其实是s的引用指向，而不是改变了String对象的值。换句话说，上面实例中 s 的值 只是 value的引用地址，并不是String内容本身！当我们执行 s = "yyg"; 语句时，Java中会创建一个新的字面量对象 "yyg"，而原来的 "yiyige" 字面量对象依然存在于内存的intern缓存池中。在Java中，因为数组也是对象，所以value中存储的也只是一个引用，它指向一个真正的数组对象。在执行了String s = "yiyige"; 这句代码之后，真正的内存布局应该是下图这样的：



因为value是String封装的字符数组，value中的所有字符都属于String这个对象。由于value是private的，且没有提供setValue等公共方法来修改这个value值，所以在String类的外部是无法修改value值的，也就是说一旦初始化就不能被修改。此外，value变量是final的，也就是说在String类内部，一旦这个值初始化了，value这个变量所引用的地址就不会改变了，即一直引用同一个对象。正是基于这一层，所以说String对象是不可变的对象。但其实value所引用对象的内容完全可以发生改变，我们可以利用反射来消除String类对象的不可变特性。

所以String的不可变性，指的是value在栈中的引用地址不可变，而不是说常量池中array本身的数据元素不可变！

而String对象的改变实际上是通过内存地址的“断开-连接”变化来完成的，这个过程中原字符串中的内容并没有任何的改变。String s = "yiyige"; 和 s = "yyg"; 实质上是开辟了2个内存空间，s只是由原来指向 "yiyige" 变为指向 "yyg" 而已，而其原来的字符串内容，是没有改变的，如下图所示。



因此，我们在以后的开发中，如果要经常修改字符串的内容，请尽量少用String，因为字符串的指向“断开-连接”会大大降低性能，建议使用：StringBuilder、StringBuffer。

那么String一定不可变吗？有没有办法让String真的可变呢？我们继续往下学习！

三. String真的不可变吗？

1. 实验案例

我在前面的章节中给大家说，String的不可变，其实指的是String类中value属性在栈中的引用地址不可变，而不是说常量池中array本身的数据元素不可变！也就是说String字符串的内容其实是可变的！那怎么实现呢？利用反射就可以实现，我们通过一个案例来证明一下。

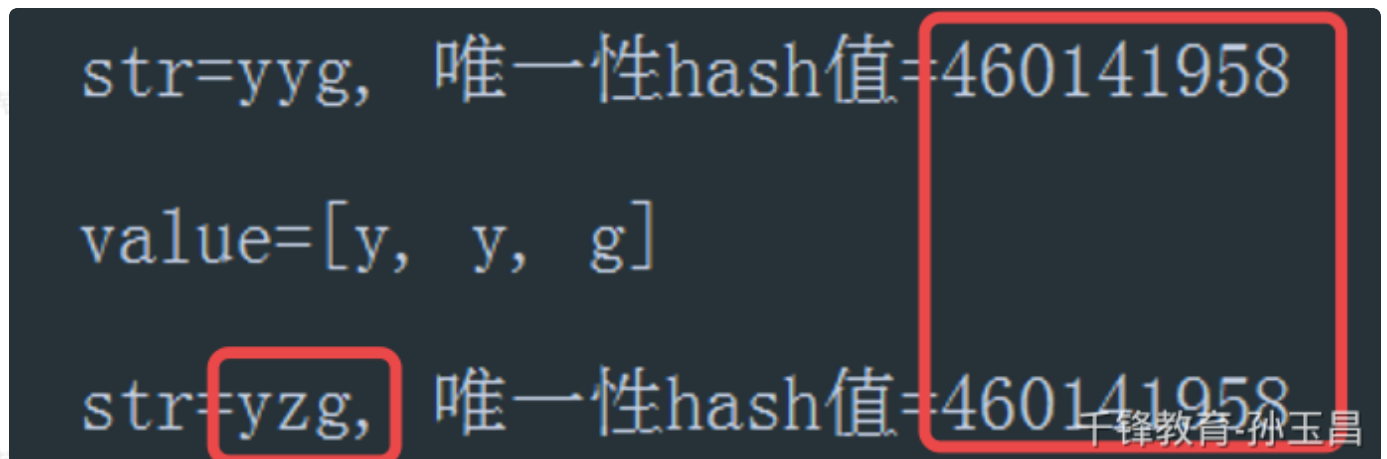
```

1 try {
2     String str = "yyg";
3     System.out.println("str=" + str + ", 唯一性hash值=" + System.identityHa
        shCode(str));
4
5     Class stringClass = str.getClass();
6     //获取String类中的value属性
7     Field field = stringClass.getDeclaredField("value");
8     //设置私有成员的可访问性,进行暴力反射
9     field.setAccessible(true);
10    //获取value数组中的内容
11    char[] value = (char[]) field.get(str);
12    System.out.println("value=" + Arrays.toString(value));
13
14    value[1] = 'z';
15    System.out.println("str=" + str + ", 唯一性hash值=" + System.identityHa
        shCode(str));
16 } catch (NoSuchFieldException | IllegalAccessException e) {
17     e.printStackTrace();
18 }

```

2. 结果剖析

上面案例的执行结果如下图所示：



```

str=yyg, 唯一性hash值=460141958
value=[y, y, g]
str=yzg, 唯一性hash值=460141958

```

我们可以看到，String字符串的字符数组可以通过反射进行修改，导致字符串的“内容”真的发生了变化！并且我们又利用底层的java.lang.System#identityHashCode()方法(不管是否重写了hashCode方法)获取了对象的唯一哈希值，该方法获取的hash值与hashCode()方法是一样的。我们可以看到两个字符串的唯一性hash值是一样的，证明字符串引用地址没有发生改变！所以在这里，我们并不是像之前那样创建了一个新的String字符串，而是真的改变了String的内容。这个代

码案例进一步说明，**String类的不可变指的是其value属性在栈中的引用地址不可变，而不是说常量池中array本身的数据元素不可变！也就是说String字符串的内容其实是可变的！**

四. 结语

String作为Java中使用最为广泛的一个类，之所以设计为不可变，主要是出于效率与安全性方面考虑。这种设计有优点，也有缺点。

1. 不可变性的优点

1. **只有当字符串是不可变的，字符串池才有可能实现。**字符串池的实现可以在运行时节约很多heap空间，因为不同的字符串引用都可以指向池中的同一个字符串。但如果字符串是可变的，如果一个引用变量改变了字符串的值，那么其它指向这个值的变量内容也会跟着一起改变。
2. **如果字符串是可变的，那么可能会引起很严重的安全问题。**譬如，数据库的用户名、密码都是以字符串的形式传入数据库，以获得数据库的连接；或者在socket编程中，主机名和端口都是以字符串的形式传入。因为字符串是不可变的，所以它的值是不可改变的，否则黑客们可以钻到空子，改变字符串指向的对象值，造成安全漏洞。
3. **因为字符串是不可变的，在物理上是绝对的线程安全，所以同一个字符串实例可以被多个线程共享。**由于不可变对象不可能被修改，因此能够在多线程中被任意自由访问而不导致线程安全问题，不需要多余的同步操作。即在并发场景下，多个线程同时读一个资源，并不会引发竞态条件，只有对资源进行读写才有危险。不可变对象不能被写，所以线程安全。
4. **类加载器要用到字符串，不可变性也提供了安全性，以便正确的类可以被加载。**譬如你想加载java.sql.Connection类，而这个值被改成了myhacked.Connection，那么会对你的数据库造成不可知的破坏。
5. **因为字符串是不可变的，所以在字符串对象创建的时候hashCode()就被执行并把执行结果缓存了，不需要重新计算。这就使得字符串很适合作为Map中的键，**所以字符串的处理速度要快过其它的键对象，这就是HashMap中的键往往都使用字符串的原因，当我们需要频繁读取访问任意键值时，能够节省很多的CPU计算开销。
6. **String的不可变性会提高执行性能和效率，基于String不可变，**我们就可以用缓存池将String对象缓存起来，同时把一个String对象的地址赋值给多个String引用，这样可以安全保证多个变量共享同一个对象。因此，构造一万个string s = "xyz"，实际上得到都是同一个字符串对象，避免了很多不必要的空间开销。

2. 不可变性的缺点

- **丧失了部分灵活性**。我们平时使用的大部分都是可变对象，比如内容变化时，只需要利用 `setValue()` 更新一下就可以了，不需要重新创建一个对象，但是 `String` 很难做到这一点。当然，我们完全可以使用 `StringBuilder` 来弥补这个缺点。
- **脆弱的不可变性，`String` 其实可以利用 JNI 或反射来改变其不可变性。**

另外，关于 `String` 源码的解读，及不可变性的相关面试题，壹哥在自己的高薪面试题专栏中有过非常详细地讲解，如果你想了解更多，请参考如下链接：

[《高薪程序员&面试题精讲系列07之说说String为什么不可变及String底层原理？》](#)

五. 今日作业

理解 `String` 不可变的原理及优缺点，并通过代码来验证 `String` 的不可变性。