

# 1、String的基本特性

- String:字符串 使用一对""引起来表示。
- String声明为final的，不可被继承
- String实现了Serializable接口：表示字符串是支持序列化的。  
实现了Comparable接口：表示String可以比较大小
- String在jdk8及以前内部定义了final char[] value用于存储字符串数据。jdk9时改为byte[]

jdk8到jdk9由 char[]改为byte[] （重大改变）

动机 (<https://openjdk.org/jeps/254>)

## String存储结构变更



结论：String 再也不用 char[] 来存储啦，改成了 byte[] 加上编码标记，节约了一些空间。

```
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence {
    @Stable
    private final byte[] value;
}
```

那StringBuffer 和 StringBuilder 是否仍无动于衷呢？

String-related classes such as AbstractStringBuilder, StringBuilder, and StringBuffer will be updated to **use the same representation**, as will the HotSpot VM's intrinsic(固有的、内置的) string operations.

让天下没有难学的技术

- `String`:代表不可变的字符序列。简称：不可变性。
  - 当对字符串重新赋值时，需要重写指定内存区域赋值，不能使用原有的value进行赋值。
  - 当对现有的字符串进行连接操作时，也需要重新指定内存区域赋值，不能使用原有的value进行赋值。
  - 当调用`String`的`replace()`方法修改指定字符或字符串时，也需要重新指定内存区域赋值，不能使用原有的value进行赋值。
- 通过字面量的方式（区别于`new`）给一个字符串赋值，此时的字符串值声明在字符串常量池中。

- 字符串常量池中是不会存储相同内容的字符串的。

- `String`的`String Pool`是一个固定大小的`Hashtable`，默认值大小长度是1009。如果放进`String Pool`的`String`非常多，就会造成Hash冲突严重，从而导致链表会很长，而链表长了后直接会造成的影响就是当调用`String.intern`时性能会大幅下降。
- 使用`-XX:StringTableSize`可设置`StringTable`的长度
- 在jdk6中`StringTable`是固定的，就是1009的长度，所以如果常量池中的字符串过多就会导致效率下降很快。`StringTableSize`设置没有要求
- 在jdk7中，`StringTable`的长度默认值是60013
- Jdk8开始， 设置`StringTable`的长度的话，1009是可设置的最小值。

`StringTable`大小变大之后，虽然占用的内存多一些，但是可以提高查询的效率。

## 2、String的内存分配

- 在Java语言中有8种基本数据类型和一种比较特殊的类型String。这些类型为了使它们在运行过程中速度更快、更节省内存，都提供了一种常量池的概念。
- 常量池就类似一个Java系统级别提供的缓存。8种基本数据类型的常量池都是系统协调的，String类型的常量池比较特殊。它的主要使用方法有两种。

- 直接使用双引号声明出来的String对象会直接存储在常量池中。
  - ✓ 比如： `String info = "atguigu.com";`
- 如果不是用双引号声明的String对象，可以使用String提供的 `intern()` 方法。这个后面重点谈

- Java 6及以前，字符串常量池存放在永久代。
- Java 7 中 Oracle<sup>1</sup> 的工程师对字符串池的逻辑做了很大的改变，即将字符串常量池的位置调整到Java堆内。
  - 所有的字符串都保存在堆（Heap）中，和其他普通对象一样，这样可以让你在进行调优应用时仅需要调整堆大小就可以了。
  - 字符串常量池概念原本使用得比较多，但是这个改动使得我们有足够的理由让我们重新考虑在Java 7 中使用 `String.intern()`。
- Java8元空间，字符串常量在堆

### 3、字符串拼接操作

1. 常量与常量的拼接结果在常量池，原理是编译期优化
2. 常量池中不会存在相同内容的常量。
3. 只要其中有一个是变量，结果就在堆中。变量拼接的原理是StringBuilder
4. 如果拼接的结果调用 `intern()` 方法，则主动将常量池中还没有的字符串对象放入池中，并返回此对象地址。

因此，字符串拼接操作不一定使用的是StringBuilder

如果拼接符号两边为字符串常量或常量引用，则仍然使用编译期优化

如果拼接符号两边为变量，则使用StringBuilder进行拼接操作

**针对于final修饰类、方法、基本数据类型、引用数据类型的类的结构时，能使用final的话，建议使用final**

```
1 package com.qy.STR;
2
3 /**
4  * @author 千祎来了
5  * @date 2022-06-16 15:28
6  */
7 public class Demo2 {
8
9     public void test1() {
10         String str1 = "a" + "b" + "c";
11         String str2 = "abc";
12
13         /**
14          * 在编译的时候，直接编译为了：
15          * String str1 = "abc";
16          * String str2 = "abc";
17          */
18         System.out.println(str1 == str2); // true
19     }
20
21     public void test2() {
22         String s1 = "javaEE";
23         String s2 = "hadoop";
24
25         String s3 = "javaEEhadoop";
26         String s4 = "javaEE" + "hadoop"; // 编译期优化
27         // 如果拼接符号前后出现了变量，则相当于在堆空间中new String()，具体内容为拼接的结果。
28         String s5 = s1 + "hadoop";
29         String s6 = "javaEE" + s2;
30         String s7 = s1 + s2;
31         System.out.println(System.identityHashCode(s1));
32         System.out.println(System.identityHashCode(s2));
33         System.out.println(System.identityHashCode(s3));
34         System.out.println(System.identityHashCode(s4));
35         System.out.println(System.identityHashCode(s5));
36         System.out.println(System.identityHashCode(s6));
```

```

37 System.out.println(System.identityHashCode(s7));
38
39 String s8 = s3.intern();
40 System.out.println(System.identityHashCode(s8));
41 }
42
43 public void test3() {
44     String s1 = "a";
45     String s2 = "b";
46     /**
47      * s1+s2的执行细节为:
48      * 1、StringBuilder s = new StringBuilder();
49      * 2、s.append("a");
50      * 3、s.append("b");
51      * 4、s.toString() --->类似于new String("ab")
52      */
53     String s3 = s1 + s2;
54     String s4 = "ab";
55     System.out.println(s3 == s4); // false
56
57     final String str1 = "a", str2 = "b";
58     String str3 = "ab";
59     String str4 = str1 + str2;
60     System.out.println(str3 == str4); // str1和str2已经不是变量了，所以没有使用StringBuilder拼接
61
62
63 }
64 public static void main(String[] args) {
65     Demo2 demo2 = new Demo2();
66     demo2.test3();
67 }
68 }

```

## 4、intern()的使用



如果不是用双引号声明的String对象, 可以使用String提供的intern方法: intern方法会从字符串常量池中查询当前字符串是否存在, 若不存在就会将当前字符串放入常量池中。

- 比如: `String myInfo = new String("I love atguigu").intern();`

也就是说, 如果在任意字符串上调用String.intern方法, 那么其返回结果所指向的那个类实例, 必须和直接以常量形式出现的字符串实例完全相同。因此, 下列表达式的值必定是true:

```
("a" + "b" + "c").intern() == "abc"
```

通俗点讲, Interned String就是确保字符串在内存里只有一份拷贝, 这样可以节约内存空间, 加快字符串操作任务的执行速度。注意, 这个值会被存放在字符串内部池(String Intern Pool)。

## 关于new String()创建的对象个数

```
1 package com.qy.STR;
2
3 /**
4  * @author 千祎来了
5  * @date 2022-06-16 16:23
6  *
7  * 题目:
8  * new String("ab")会创建几个对象? 看字节码文件, 是两个。
9  * 对象1: new关键字在堆空间创建
10  * 对象2: 字符串常量池中的对象"ab" 字节码指令: ldc
11  *
12  * 思考:
13  * new String("a") + new String("b")呢?
14  * 对象1: new StringBuilder()
15  * 对象2: new String("a")
16  * 对象3: 常量池中的"a"
17  * 对象4: new String("b")
18  * 对象5: 常量池中的"b"
19  * 深入剖析: StringBuilder的toString()方法:
20  * 对象6: new String("ab")
21  * SringBuilder的toString()不同于其他的toString(), 这里在字符串常量池中并没有生成"ab"
22  */
23 public class Demo3 {
24     public static void main(String[] args) {
```

```

25 // String str = new String("ab");
26 String str2 = new String("a") + new String("b");
27 }
28 }

```

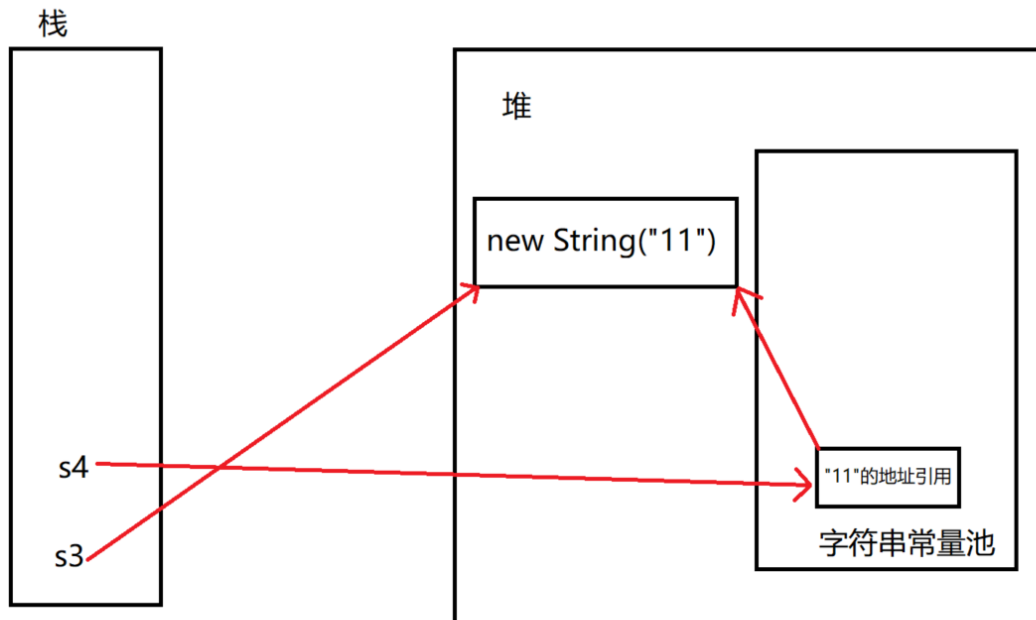
## 关于intern()的面试难题 ★

关于以下代码在jdk8中如何存储如下图所示：

```

1 String s3 = new String("1") + new String("1");
2 s3.intern(); // 在字符串常量池中生成"11"
3 String s4 = "11"; // s4指向常量池中的"1"

```



```

1 package com.qy.STR;
2
3 /**
4  * @author 千祎来了
5  * @date 2022-06-16 16:32
6  */
7 public class Demo4 {
8     public static void main(String[] args) {
9
10    /**

```

```

11  * 创建了两个对象：1、堆空间的“1” 2、常量池的“1”
12  */
13  String s = new String("1");
14  s.intern(); // 把"1"放到常量池 （放之前，常量池已经有"1"了）
15  String s2 = "1";
16  System.out.println(s == s2); // jdk6:false jdk7/8:false
17
18  /**
19  * s3变量记录的地址为：new String("11")
20  * 1、堆空间的“1”
21  * 2、常量池的“1”
22  * 3、堆空间的“1” （StringBuilder.toString()没有创建常量池对象）
23  */
24  String s3 = new String("1") + new String("1");
25  // 执行完上一行代码后，字符串常量池中并不存在对象"11"
26  /**
27  * s3.intern()如何理解：
28  * jdk6: 创建了一个新的对象"11"，也就有新的地址
29  * jdk7/8: 由于s3的创建，堆中已经有一个new String("11")的对象了，调用
intern()之后，为了节省空间，
30  * 直接将堆中的new String("11")这个对象的地址放在了字符串常量池中
31  */
32  s3.intern(); // 在字符串常量池中生成"11"
33
34  String s4 = "11"; // s4指向常量池中的"11"
35  System.out.println(s3 == s4); // jdk6:false jdk7/8:true
36  }
37  }

```

## 总结

常量池在下图中简写为串池



总结String的intern()的使用:

- jdk1.6中, 将这个字符串对象尝试放入串池。
  - 如果串池中有, 则并不会放入。返回已有的串池中的对象的地址
  - 如果没有, 会把此对象复制一份, 放入串池, 并返回串池中的对象地址
- Jdk1.7起, 将这个字符串对象尝试放入串池。
  - 如果串池中有, 则并不会放入。返回已有的串池中的对象的地址
  - 如果没有, 则会把对象的引用地址复制一份, 放入串池, 并返回串池中的引用地址

**当需要大量使用字符串时, 尤其存在大量重复的字符串, 使用intern()可以节省很多的内存空间**

```
1 new String(str)
2 new String(str).intern() // 更节省空间
```