# 简述

String 类也是java.lang 包下的一个类，算是日常编码中最常用的一个类了

String 类是char的一个数组

# String类定义对象的说明

String 类是final类型的常量类，不能被继承，**一旦一个String对象被创建，包含在这个对象中的字符序列是不可改变的，包括该类后续的**
**该对象被销毁，这是需要特别注意的（该类的一些方法看似改变了字符串，其实内部都是创建一个新的字符串）**。接着实现了 Serializable接
现了 Comparable 接口，用于比较两个字符串的大小（按顺序比较单个字符的ASCII码）；最后实现了 CharSequence 接口，表示是一个有

# 字段属性

```
/**用来存储字符串  */
private final char value[];

 /** 缓存字符串的哈希码 */
private int hash; // Default to 0
```

# 构造方法

- String()
- String(String)
- String(char[])
- String(char[], int, int)
- String(int[], int, int)
- ~~String(byte[], int, int, int)~~
- ~~String(byte[], int)~~
- String(byte[], int, int, String)
- String(byte[], int, int, Charset)
- String(byte[], String)
- String(byte[], Charset)
- String(byte[], int, int)
- String(byte[])
- String(StringBuffer)
- String(StringBuilder)
- String(char[], boolean)

String 类的构造方法很多。可以通过初始化一个字符串，或者字符数组，或者字节数组等等来创建一个 String 对象

```
String str1 = "abc";
String str2 = new String("abc");
String str3 = new String(new char[]{'a','b','c'});
```

# 其他方法

## equals

String 类重写了Object的 equals 方法，比较的是组成字符串的每一个字符是否相同，如果都相同则返回true，否则返回false

## hashCode

String 类的 hashCode 算法很简单，主要就是中间的 for 循环，计算公式如下：

```
s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
```

s 数组即源码中的 val 数组，也就是构成字符串的字符数组。这里有个数字 **31**，为什么选择31作为乘积因子，而且没有用一个常量来声
①、31是一个不大不小的质数，是作为 hashCode 乘子的优选质数之一。
②、31可以被 JVM 优化，`31 * i = (i << 5) – i`。因为移位运算比乘法运行更快更省性能

详细解释（String hashCode 方法为什么选择数字31作为乘子）：https://www.cnblogs.com/nullllun/p/8350178.html

## charAt

通过传入的索引（数组下标），返回指定索引的单个字符

## compareTo(String anotherString) 和 compareToIgnoreCase(String str) 方法

```
public int compareTo(String anotherString) {
    int len1 = value.length;
    int len2 = anotherString.value.length;
    int lim = Math.min(len1, len2);
    char v1[] = value;
    char v2[] = anotherString.value;

    int k = 0;
```

```
    while (k < lim) {
        char c1 = v1[k];
        char c2 = v2[k];
        if (c1 != c2) {
            return c1 - c2;
        }
        k++;
    }
    return len1 - len2;
}
```

按字母顺序比较两个字符串，是基于字符串中每个字符的 Unicode 值。当两个字符串某个位置的字符不同时，返回的是这一位置的字符
同时，返回两个字符串长度之差。

compareToIgnoreCase() 方法在 compareTo 方法的基础上忽略大小写，我们知道大写字母是比小写字母的Unicode值小32的，底层实
成小写进行比较

### concat(String str)

该方法是将指定的字符串连接到此字符串的末尾

首先判断要拼接的字符串长度是否为0，如果为0，则直接返回原字符串。如果不为0，则通过 Arrays 工具类（后面会详细介绍这个工具
数组，长度为原字符串和要拼接的字符串之和，前面填充原字符串，后面为空。接着在通过 getChars 方法将要拼接的字符串放入新字符串

**注意**：返回值是 new String(buf, true)，也就是重新通过 new 关键字创建了一个新的字符串，原字符串是不变的。这也是前面我们说的
**个对象中的字符序列是不可改变的。**

### indexOf、split、substring、replace（略）

## 常量池

java运行时会维护一个String Pool（String池），也叫“字符串缓冲区”。String池用来存放运行时中产生的各种字符串，并且池中的字符

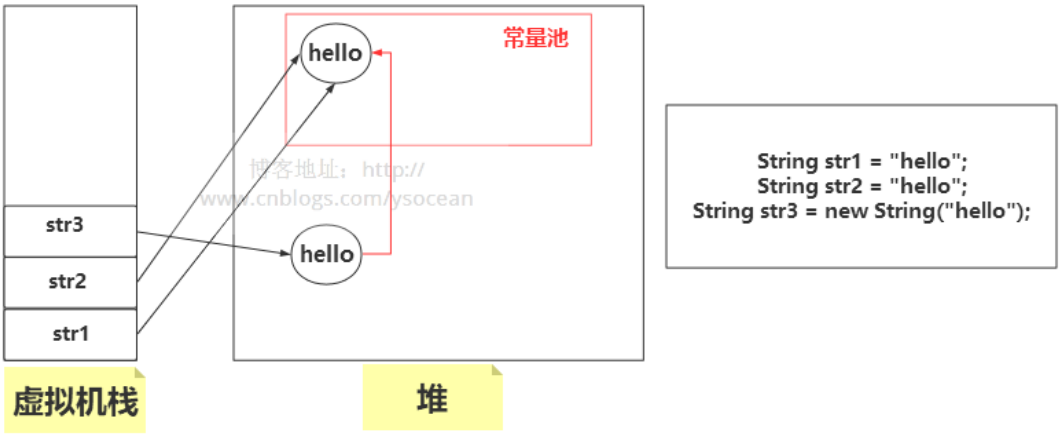
1. 纯字符串或者纯字符串（常量）拼接字符串会先在字符串池中找，看是否有相等的对象，没有的话就在字符串池创建该对象；有的话
象

2. new关键字创建时，直接在堆中创建一个新对象，变量所引用的都是这个新对象的地址，但是如果通过new关键字创建的字符串内容
常量池的对应字符；但是反过来，如果通过new关键字创建的字符串对象在常量池中没有，那么通过new关键词创建的字符串对象是不

3. 使用包含变量表达式来创建String对象，则不仅会检查维护字符串池，还会在堆区创建这个对象，最后是指向堆内存的对象

```
String str1 = "hello";
String str2 = "hello";
String str3 = new String("hello");
System.out.println(str1==str2);//true
System.out.println(str1==str3);//fasle
System.out.println(str2==str3);//fasle
System.out.println(str1.equals(str2));//true
System.out.println(str1.equals(str3));//true
System.out.println(str2.equals(str3));//true
```

对于上面的情况，首先 String str1 = "hello"，会先到常量池中检查是否有“hello”的存在，发现是没有的，于是在常量池中创建“hello”对象，二个字面量 String str2 = "hello"，在常量池中检测到该对象了，直接将引用赋值给str2；第三个是通过new关键字创建的对象，常量池中有后在堆中创建该对象后，将堆中对象的引用赋值给str3，再将该对象指向常量池。如下图所示：



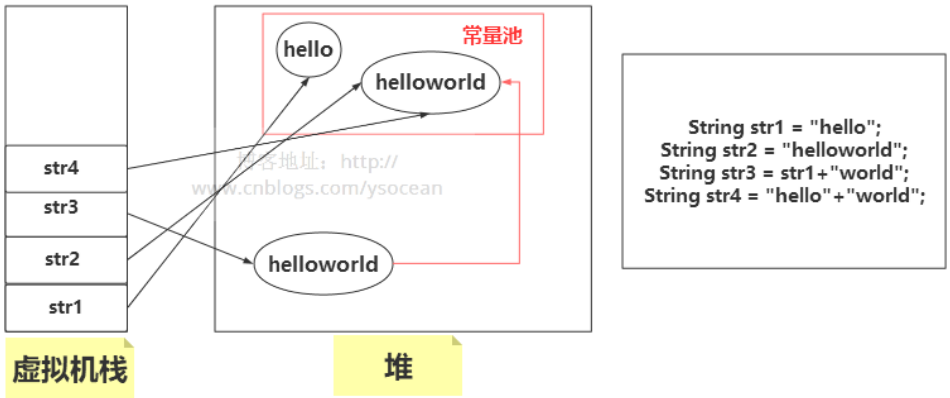

**注意**：看上图红色的箭头，通过 new 关键字创建的字符串对象，如果常量池中存在了，会将堆中创建的对象指向常量池的引用。我们可验证

### 使用包含变量表达式创建对象

```
String str1 = "hello";
String str2 = "helloworld";
String str3 = str1+"world";//编译器不能确定为常量(会在堆区创建一个String对象)
String str4 = "hello"+"world";//编译器确定为常量，直接到常量池中引用

System.out.println(str2==str3);//fasle
System.out.println(str2==str4);//true
System.out.println(str3==str4);//fasle
```

str3 由于含有变量str1，编译器不能确定是常量，会在堆区中创建一个String对象。而str4是两个常量相加，直接引用常量池中的对象即可




## intern() 方法

这是一个本地方法：

```
public native String intern();
```

当调用intern方法时，如果池中已经包含一个与该String确定的字符串相同equals(Object)的字符串，则返回该字符串。否则，将此象的引用

```
String str1 = "hello";//字面量 只会在常量池中创建对象
String str2 = str1.intern();
System.out.println(str1==str2);//true

String str3 = new String("world");//new 关键字只会在堆中创建对象
String str4 = str3.intern();
System.out.println(str3 == str4);//false

String str5 = str1 + str2;//变量拼接的字符串，会在常量池中和堆中都创建对象
 String str6 = str5.intern();//这里由于池中已经有对象了，直接返回的是对象本身，也就是堆中的对象
 System.out.println(str5 == str6);//true

 String str7 = "hello1" + "world1";//常量拼接的字符串，只会在常量池中创建对象
 String str8 = str7.intern();
 System.out.println(str7 == str8);//true
```

# String真的不可变?

每个字符串都是由许多单个字符组成的，我们知道其源码是由 char[] value 字符数组构成

value 被 final 修饰，只能保证引用不被改变，但是 value 所指向的堆中的数组，才是真实的数据，只要能够操作堆中的数组，依旧能改变数

而且 value 是基本类型构成，那么一定是可变的，即使被声明为 private，我们也可以通过反射来改变

```
String str = "vae";
//打印原字符串
System.out.println(str);//vae
//获取String类中的value字段
Field fieldStr = String.class.getDeclaredField("value");
//因为value是private声明的，这里修改其访问权限
fieldStr.setAccessible(true);
//获取str对象上的value属性的值
char[] value = (char[]) fieldStr.get(str);
 //将第一个字符修改为 V(小写改大写)
 value[0] = 'V';
 //打印修改之后的字符串
 System.out.println(str);//Vae
```

**结论：通过前后两次打印的结果，我们可以看到 String 被改变了，但是在代码里，几乎不会使用反射的机制去操作 String 字符串，所以，**

# 为什么String要被设计成不可变?

String 类为什么要这样设计成不可变呢？我们可以从性能以及安全方面来考虑

## 安全

- 引发安全问题，譬如，数据库的用户名、密码都是以字符串的形式传入来获得数据库的连接，或者在socket编程中，主机名和端口都不可变的，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的值，造成安全漏洞。

- 保证线程安全，在并发场景下，多个线程同时读写资源时，会引竞态条件，由于 String 是不可变的，不会引发线程的问题而保证了线

- HashCode，当 String 被创建出来的时候，hashcode也会随之被缓存，hashcode的计算与value有关，若 String 可变，那么 hashco
  等容器，他们的键值需要保证唯一性和一致性，因此，String 的不可变性使其比其他对象更适合当容器的键值

## 性能

当字符串是不可变时，字符串常量池才有意义。字符串常量池的出现，可以减少创建相同字面量的字符串，让不同的引用指向池中同一
存。若字符串可变，字符串常量池失去意义，基于常量池的String.intern()方法也失效，每次创建新的 String 将在堆内开辟出新的空间，占