1、String类及常用方法

1.String的特性

String:字符串，使用一对""引起来表示。

1.String声明为final的，不可被继承

2.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

3.String内部定义了final char[] value用于存储字符串数据

4. String:代表不可变的字符序列。简称：不可变性。

体现:

1. 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。

2. 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。

3. 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。

```
public class StringTest {
@Test
public void test1(){
String s1 = "abc";//字面量的定义方式
String s2 = "abc";
s1 = "hello";

System.out.println(s1 == s2);//比较s1和s2的地址值

 System.out.println(s1);//hello
 System.out.println(s2);//abc

 System.out.println("*****************");

 String s3 = "abc";
 s3 += "def";
 System.out.println(s3);//abcdef
 System.out.println(s2);

 System.out.println("*****************");

 String s4 = "abc";
 String s5 = s4.replace('a', 'm');
 System.out.println(s4);//abc
 System.out.println(s5);//mbc
 }
 }
```

2. String实例化

1. 方式一:通过字面量的方式

通过字面量定义的方式：此时的s1和s2的数据javaEE声明在方法区中的字符串常量池中。

2. 方式二:通过new+构造器的方式

通过new + 构造器的方式:此时的s3和s4保存的地址值，是数据在堆空间中开辟空间以后对应的地址值。

3.面试题：String s = new String("abc");方式创建对象，在内存中创建了几个对象？

两个:一个是堆空间中new结构，另一个是char[]对应的常量池中的数据："abc"

```
@Test
 public void test2(){
 //通过字面量定义的方式：此时的s1和s2的数据javaEE声明在方法区中的字符串常量池中。
 String s1 = "javaEE";
 String s2 = "javaEE";
 //通过new + 构造器的方式:此时的s3和s4保存的地址值，是数据在堆空间中开辟空间以后对应的地址值。
 String s3 = new String("javaEE");
 String s4 = new String("javaEE");

  System.out.println(s1 == s2);//true
  System.out.println(s1 == s3);//false
  System.out.println(s1 == s4);//false
  System.out.println(s3 == s4);//false

  System.out.println("**********************");
  Person p1 = new Person("Tom",12);
  Person p2 = new Person("Tom",12);

  System.out.println(p1.name.equals(p2.name));//true
  System.out.println(p1.name == p2.name);//true

  p1.name = "Jerry";
  System.out.println(p2.name);//Tom
  }
```

3.总结

1.常量与常量的拼接结果在常量池。且常量池中不会存在相同内容的常量。

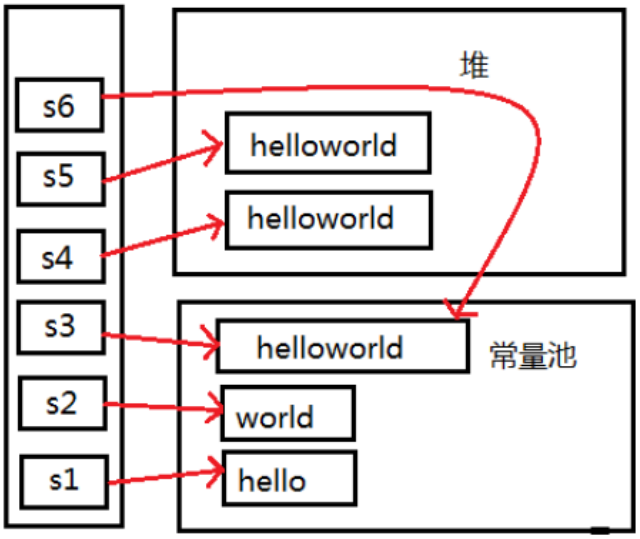
2.只要其中有一个是变量，结果就在堆中。

3.如果拼接的结果调用intern()方法，返回值就在常量池中

```
public class StringTest {
String str = new String("good");
char[] ch = { 't', 'e', 's', 't' };

public void change(String str, char ch[]) {
str = "test ok";
ch[0] = 'b';
}
public static void main(String[] args) {
 StringTest ex = new StringTest();
 ex.change(ex.str, ex.ch);
 System.out.println(ex.str);//good 不可变性
```

```
System.out.println(ex.ch);//best
}
}

@Test
public void test4(){
String s1 = "javaEEhadoop";
String s2 = "javaEE";
String s3 = s2 + "hadoop";
System.out.println(s1 == s3);//false

final String s4 = "javaEE";//s4:常量
String s5 = s4 + "hadoop";
System.out.println(s1 == s5);//true

}
```

```
String s1 = "hello";

String s2 = "world";

String s3 = "hello" + "world";

String s4 = s1 + "world";

String s5 = s1 + s2;

String s6 = (s1 + s2).intern();
System.out.println(s3==s4);//false
System.out.println(s3==s5);//false
System.out.println(s4==s5);//false
System.out.println(s3==s6);//true
```



4. String的常用方法

```
int length() 返回字符串的长度： return value length

char charAt(int index) 返回某索引处的字符 return value[index]

boolean isEmpty() 判断是否是空字符串： return value length 0

String toLowerCase() 使用默认语言环境 将 String 中的所有字符转换为小写

String toUpperCase() 使用默认语言环境 将 String 中的所有字符转换为大写

String trim() 返回字符串的副本 忽略前导空白和尾部 空白
boolean equals(Object obj) 比较字符串的 内容 是否相同

```

```
14  boolean equalsIgnoreCase(String anotherString) 与 equals 方法类似 忽略大
15 小写
16
17  String concat(String str) 将指定字符串连接到此字符串的结尾 。 等价于用" + "
18
19  int compareTo(String anotherString) 比较两个字符串的大小
20
21  String substring(int beginIndex) 返回一个新的字符串 它是此字符串的从
22  beginIndex 开始截取到最后的一个子字符串 。
23
24  String substring(int beginIndex, int endIndex) 返回一个新字符串 它是此字
25  符串从 beginIndex 开始截取到 endIndex( 不包含 的一个子字符串
26
27  boolean endsWith(String suffix) 测试此字符串是否以指定的后缀结束
28
29  boolean startsWith(String prefix) 测试此字符串是否以指定的前缀开始
30
31  boolean startsWith(String prefix, int toffset) 测试此字符串从指定索引开始的
32 子字符串是否以指定前缀开始
33
34  boolean contains(CharSequence s) 当且仅当此字符串包含指定的 char 值序列
35 时，返回 true
36
37  int indexOf(String str) 返回指定子字符串在此字符串中第一次出现处的索引
38
39  int indexOf(String str, int fromIndex) 返回指定子字符串在此字符串中第一次出
40 现处的索引，从指定
41  的索引开始
42
43  int lastIndexOf(String str) 返回指定子字符串在此字符串中最右边出现处的索引
44
45  int lastIndexOf(String str, int fromIndex) 返回指定子字符串在此字符串中最后
46 一次出现处的索引，从指定
47  的索引开始反向 搜索
48  注:
49 indexOf 和 lastIndexOf 方法如果未找到都是返回 -1
50
51  什么情况下，indexOf(str)和lastIndexOf(str)返回值相同?
52  情况一：存在唯一的一个str。情况二：不存在str
53
54  String replace(char oldChar, char newChar) 返回 一个新的字符串 它是
55 通过用 newChar 替换此字符串
56  中出现的所有 oldChar 得到的 。
57
```

```
String replace(CharSequence target, CharSequence replacement) 使用指定的字面值替换序列替换此字符串所有匹配字面值目标序列的子字符串 。

String replaceAll(String regex, String replacement) 使用给定的replacement 替换此字符串所有匹配给定的正则表达式的子字符串 。

String replaceFirst(String regex, String replacement) 使用 给定的replacement 替换此字符串匹配给定的正则表达式的第一个子字符串

boolean matches(String regex) 告知此字符串是否匹配给定的正则表达式 。

String[] split(String regex) 根据给定正则表达式的匹配拆分此字符串 。

String[] split(String regex, int limit) 根据匹配给定的正则表达式来拆分此字符串最多不超过 limit 个 如果超过了 剩下的全部都放到最后一个元素中
```

5.String和基本数据类型的转换

5.1 字符串 ==>基本数据类型、包装类

Integer 包装类的 public static int parseInt (String s)：可以将由“数字”字符组成的字符串转换为整型。

类似地 使用 java.lang 包中的 Byte 、 Short 、 Long 、 Float 、 Double 类调相应的类方法可以将由 “数字”字符 组成的字符串，转化为相应的基本数据类型。

5.2 基本数据类型、包装类==>字符串

调用 String 类的 public String valueOf int n) 可将 int 型转换为字符串相应的valueOf (byte 、 valueOf (long 、 valueOf (float 、 valueOf (doubled) 、 valueOf boolean b) 可由参数的相应类型到字符串的转换

```java
@Test
public void test1(){
String str1 = "123";
// int num = (int)str1;//错误的
int num = Integer.parseInt(str1);

String str2 = String.valueOf(num);//"123"
String str3 = num + "";

System.out.println(str1 == str3);
}
```

6.String和字符数组的转换

6.1 字符串 -->字符数组

String --> char[]:调用String的toCharArray()

public void getChars(int srcBegin, int srcEnd, char[] dst,int dstBegin 提供了将指定索引范围内的字符串存放到数组中的方法。

6.2 字符数组 -->字符串

char[] --> String:调用String的构造器

String类的构造器： String(char[]) 和 String(char[] int offset int length) 分别用字符数组中的全部字符和部分字符创建字符串对象。

```
@Test
public void test2(){
String str1 = "abc123"; //题目：  a21cb3

char[] charArray = str1.toCharArray();
for (int i = 0; i < charArray.length; i++) {
System.out.println(charArray[i]);
}

 char[] arr = new char[]{'h','e','l','l','o'};
 String str2 = new String(arr);
 System.out.println(str2);
 }
```

7.String与字节数组转换

7.1 字符串-->字节数组

编码：String --> byte[]:调用String的getBytes()

编码：字符串 -->字节 （看得懂 --->看不懂的二进制数据）

7.2 字节数组-->字符串

解码：byte[] --> String:调用String的构造器

解码：编码的逆过程，字节 --> 字符串 （看不懂的二进制数据 ---> 看得懂）

7.3 说明

解码时，要求解码使用的字符集必须与编码时使用的字符集一致，否则会出现乱码。

```
public class StringTest1 {
@Test
public void test3() throws UnsupportedEncodingException {
String str1 = "abc123中国";
byte[] bytes = str1.getBytes();//使用默认的字符集，进行编码。
System.out.println(Arrays.toString(bytes));

byte[] gbks = str1.getBytes("gbk");//使用gbk字符集进行编码。
System.out.println(Arrays.toString(gbks));

 System.out.println("************************************");

 String str2 = new String(bytes);//使用默认的字符集，进行解码。
 System.out.println(str2);
```

```
15
16  String str3 = new String(gbks);
17  System.out.println(str3);//出现乱码。原因：编码集和解码集不一致！
18
19
20  String str4 = new String(gbks, "gbk");
21  System.out.println(str4);//没有出现乱码。原因：编码集和解码集一致！
22
23
24  }
```

2. StringBuffer

2,1 String、StringBuffer、StringBuilder三者的异同？

- String:不可变的字符序列；底层使用char[]存储
- StringBuffer:可变的字符序列；线程安全的，效率低；底层使用char[]存储
- StringBuilder:可变的字符序列；jdk5.0新增的，线程不安全的，效率高；底层使用char[]存储

```
1   /*
2   对比String、StringBuffer、StringBuilder三者的效率：
3   从高到低排列：StringBuilder > StringBuffer > String
4   */
5   @Test
6   public void test3(){
7   //初始设置
8   long startTime = 0L;
9   long endTime = 0L;
10   String text = "";
11   StringBuffer buffer = new StringBuffer("");
12   StringBuilder builder = new StringBuilder("");
13   //开始对比
14   startTime = System.currentTimeMillis();
15   for (int i = 0; i < 20000; i++) {
16   buffer.append(String.valueOf(i));
17   }
18   endTime = System.currentTimeMillis();
19   System.out.println("StringBuffer的执行时间：" + (endTime - startTime));
20
21   startTime = System.currentTimeMillis();
22   for (int i = 0; i < 20000; i++) {
23   builder.append(String.valueOf(i));
24   }
25   endTime = System.currentTimeMillis();
26   System.out.println("StringBuilder的执行时间：" + (endTime - startTime));
27
```

```
startTime = System.currentTimeMillis();
for (int i = 0; i < 20000; i++) {
text = text + i;
}
endTime = System.currentTimeMillis();
System.out.println("String的执行时间：" + (endTime - startTime));
}
```

2.2 源码分析

```
String str = new String();//char[] value = new char[0];
String str1 = new String("abc");//char[] value = new char[]{'a','b','c'};

// 底层创建了一个长度是16的数组。
StringBuffer sb1 = new StringBuffer();//char[] value = new char[16];
System.out.println(sb1.length());// 0
sb1.append('a');//value[0] = 'a';
sb1.append('b');//value[1] = 'b';

//char[] value = new char["abc".length() + 16];
StringBuffer sb2 = new StringBuffer("abc");
System.out.println(sb2.length());// 3
```

2.3 问题

问题1. System.out.println(sb2.length());//3

问题2. 扩容问题:如果要添加的数据底层数组盛不下了，那就需要扩容底层的数组。默认情况下，扩容为原来容量的2倍 + 2，同时将原有数组中的元素复制到新的数组中。

2.4 指导意义

开发中建议大家使用：StringBuffer(int capacity) 或 StringBuilder(int capacity)

2.5 StringBuffer类的常用方法

```
StringBuffer
 append ( xxx)：提供了很多的 append() 方法 用于进行字符串拼接
StringBuffer
 delete (int start,int end)end)：删除指定位置的内容
StringBuffer
 replace (int start, int end, String str)str)：把 [start, 位置替换为 str
StringBuffer
 insert (int offset, xxx)xxx)：在指定位置插入 xxx
StringBuffer
 reverse ()()：把当前字符序列逆转
public in
 t indexOf (String str)
public String
```

```
14  substring ((int start,int end) 返回一个从start开始到end索引结束的左闭右开区间的子字符串
15 public int
16  length())
17 public char
18  charAt (int n)
19 public void
20  setCharAt (int n ,char ch)
```

3.StringBuilder