

String概述

基本介绍

基本介绍

- 一个字符串是由多个字符组成的一串数据(字符序列,字符数组)
- String类代表字符串,Java 程序中的所有字符串字面值(如 "abc")都作为此类的实例实现
- 在java.lang包下,是java核心类,最常用类,但是不属于基本数据类型,
- String类提供了字符串表示、比较、查找、截取、大小写转换等各种针对字符串的操作是引用类型

构造方法

- ""空字符串
- byte[] ----> String
- char[] ----> String

```
//空字符串 "" null
public String()

//利用字节数组,创建字节数组所表示的字符串
// 1. 字符 -> 数值形式 'a' -> 97
// 2. 所以可以用多个字节值,表示多个字符-->即字符序列 public
String(byte[] bytes)

//利用字节数组的一部分,创建字符序列,从byte数组的offset开始的length个字节值
public String(byte[] bytes,int offset,int length)

//利用一个字符数组创建字符串,代表的字符序列
public String(char[] value)

// 创建value字符数组中,从第offset位置开始的count个字符,所代表的字符串对象
public String(char[] value,int offset,int count)

//知道即可
public String(String original)
```

Demo

```
package _15string.com.cskaoyan._01introduction;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/18 9:38
 */

public class Demo1 {
    public static void main(String[] args) {
        // //空字符串 "" null
    }
}
```

```

// public String()
String s = new String();
System.out.println("s = " + s);
// //利用字节数组，创建字节数组所表示的字符串
// // 1. 字符 -> 数值形式 'a' -> 97
// // 2. 所以可以用多个字节值，表示多个字符-->即字符序列 public
// String(byte[] bytes)
byte[] bytes = {97, 98, 99};
String s1 = new String(bytes);
System.out.println("s1 = " + s1);

// //利用字节数组的一部分，创建字符序列，从byte数组的offset开始的length个字节值
// public String(byte[] bytes,int offset,int length)
String s2 = new String(bytes, 1, 1);
System.out.println("s2 = " + s2);

// //利用一个字符数组创建字符数组，代表的字符序列
// public String(char[] value)
char[] chars = {'h', 'e', 'l'};
String s3 = new String(chars);
System.out.println("s3 = " + s3);

// // 创建value字符数组中，从第offset位置开始的count个字符，所代表的字符串对象
// public String(char[] value,int offset,int count)
//
// //知道即可
// public String(String original)
String s4 = new String("abc");
System.out.println("s4 = " + s4);
}
}

```

String特点(重点)

String对象不可变

对象一旦被创建后，对象所有的状态及属性在其生命周期内不会发生任何变化。

1.请键盘录入一个任意字符串s，并用一个temp字符串引用也指向它
这个时候修改temp字符串的内容，请问s字符串的内容会随之改变吗？

```

package _15string.com.cskaoyan._02feature;

import java.util.Scanner;

/**
 * @description:
 * @author: 景天

```

* @date: 2022/7/20 10:27

**/

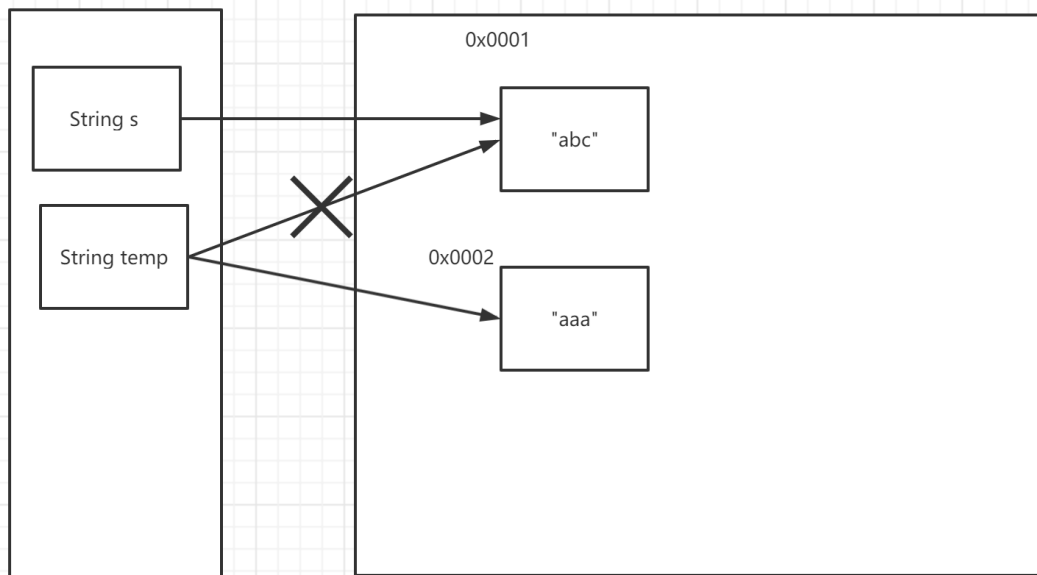
/*

String 对象不可变

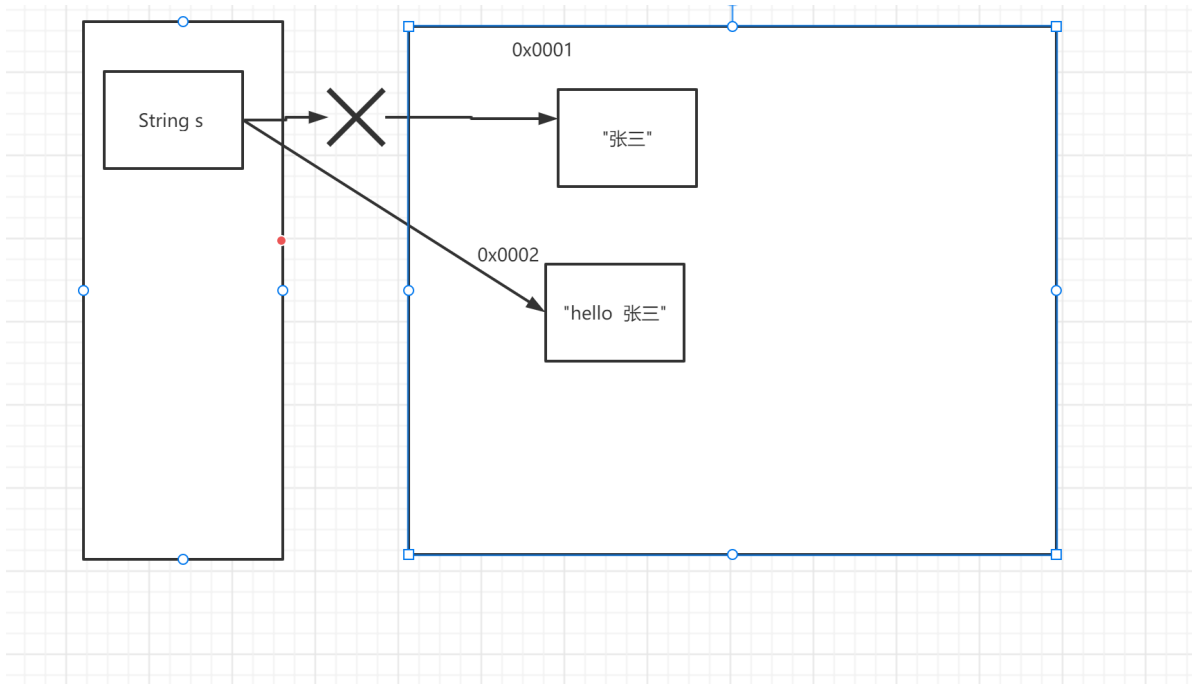
1. 请键盘录入一个任意字符串s，并用一个temp字符串引用也指向它
这个时候修改temp字符串的内容，请问s字符串的内容会随之改变吗？

*/

```
public class Demo1 {  
    public static void main(String[] args) {  
        // 创建Scanner对象  
        Scanner scanner = new Scanner(System.in);  
        System.out.println("请输入:");  
        // 键盘接收 nextLine  
        String s = scanner.nextLine();  
  
        // temp字符串引用也指向它  
        String temp = s;  
        System.out.println("temp = " + temp);  
        // 修改temp字符串的内容  
        temp = "aaa";  
        // 打印  
        System.out.println("temp = " + temp);  
        System.out.println("s = " + s);  
    }  
}
```



```
String s = "张三";
System.out.println("s = " + s);
s = "hello 张三";
System.out.println("s = " + s);
```



不可变的原因与本质

String是一个final类,代表不可变的字符序列

字符串是常量,用双引号引起来,他们的值在创建之后不可更改

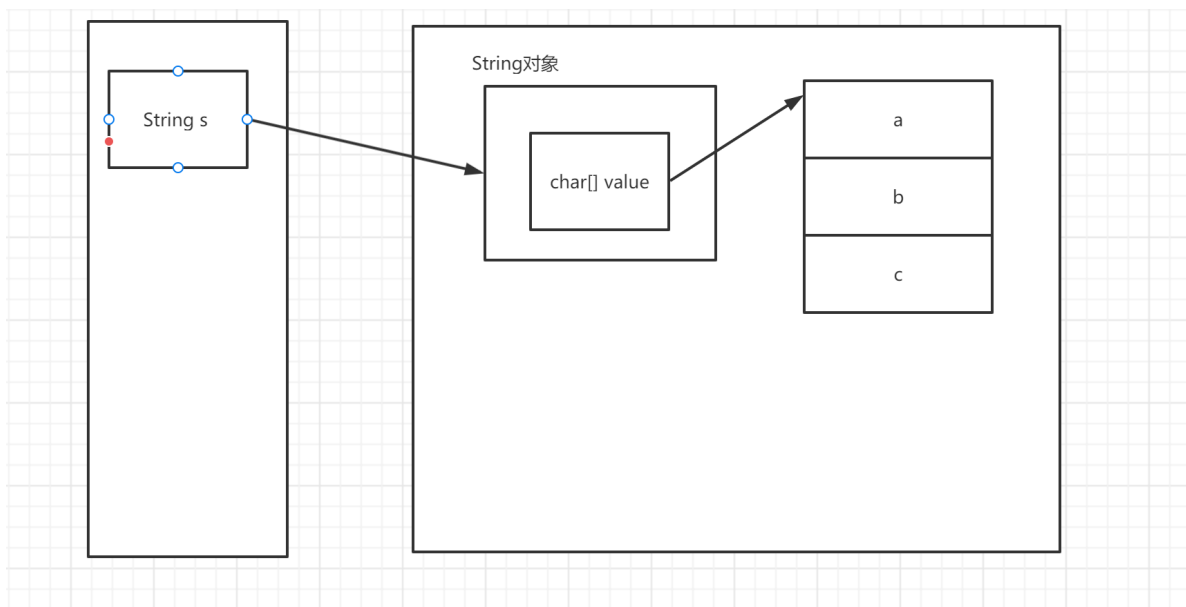
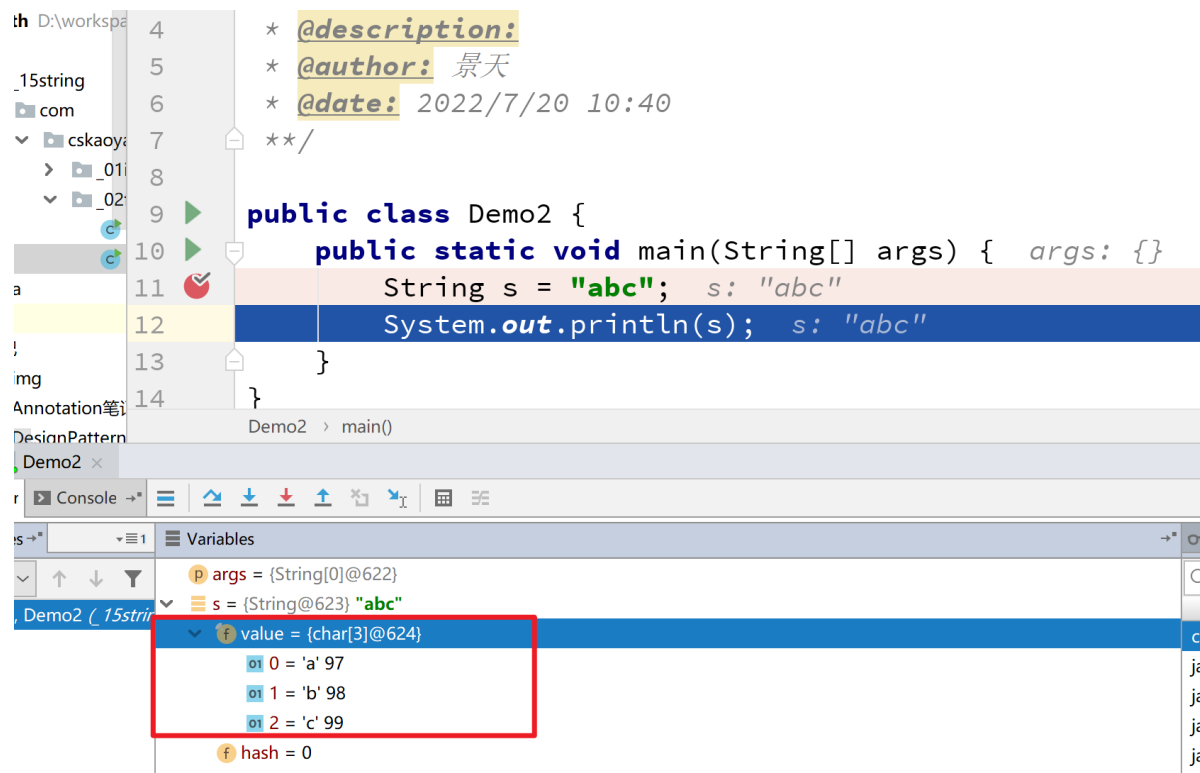
String对象的内容是存储在字符数组value[]中的

```
public final class String 意味着不能被继承
    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;

    /**
     * Class String is special cased within the Serialization Stream Protocol.
     *
     * A String instance is written into an ObjectOutputStream according to
     * <a href="{@docRoot}/../platform/serialization/spec/output.html">
     * Object Serialization Specification, Section 6.2, "Stream Elements"</a>
     */
```



字符串常量池

字符串的分配和其他对象分配一样，是需要消耗高昂的时间和空间的，而且字符串使用的非常多

JVM为了提高性能和减少内存的开销，在实例化字符串对象的时候进行了一些优化：

使用字符串常量池。

首先要明确，Java的双引号引起的字面值常量字符串，它们都是对象。这些对象比较特殊，程序在编译时期就能确定它们的值

每当创建字符串常量对象时，JVM会首先检查字符串常量池，如果该字符串对象引用已经存在常量池中，那么就直接返回常量池中的实例引用。如果字符串对象引用不存在于常量池中，就会实例化该字符串并且将其引用放到常量池中。

```

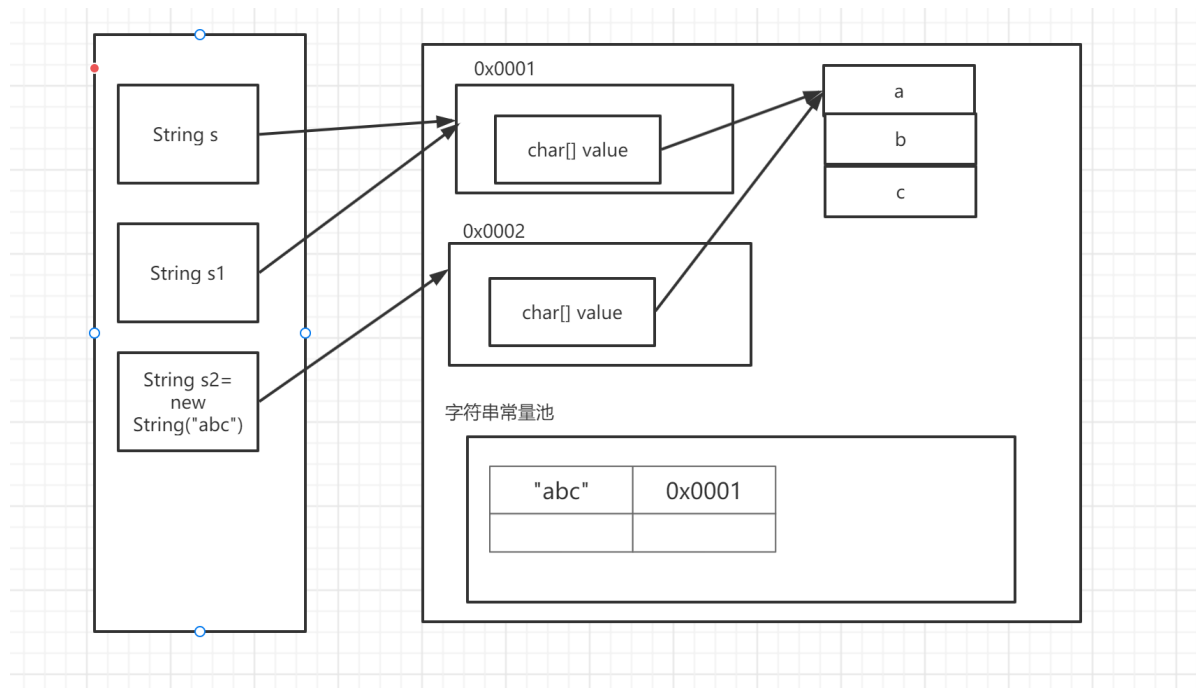
11  */
12  public class Demo3 {
13      public static void main(String[] args) { args: {}
14          String s1 = "abc"; s1: "abc"
15          String s2 = "abc"; s2: "abc"
16          String s3 = new String(original: "abc"); s3: "abc"
17          System.out.println(s1 == s2); // true s1: "abc" s2: "abc"
18          System.out.println(s1 == s3); // false

```

Variables

- args = {String[0]@622}
- s1 = {String@623} "abc"
 - value = {char[3]@624}
 - hash = 0
- s2 = {String@623} "abc"
 - value = {char[3]@624}
 - hash = 0
- s3 = {String@626} "abc"
 - value = {char[3]@624}
 - hash = 0

| Class | Count | Diff |
|----------------------------|-------|------|
| java.lang.String | 2778 | +1 |
| char[] | 2926 | 0 |
| java.util.TreeMap\$Entry | 791 | 0 |
| java.lang.Class | 692 | 0 |
| java.lang.Object[] | 650 | 0 |
| int[] | 353 | 0 |
| sun.misc.FDBigInteger | 341 | 0 |
| java.util.Hashtable\$Entry | 282 | 0 |
| java.lang.Integer | 256 | 0 |



String两种实例化方式

- 直接赋值 String s = "abc";(常用)
- 构造方法 String s = new String("abc");

总结:

```

String s = "abc"; // 创建一个String对象 加入常量池
String s1 = new String("abc"); //只创建1个

```

```

String s1 = new String("abc"); //只创建2个."abc"创建出来,加入常量池, 又new出来一个新的String对象
String s = "abc"; // 不创建 ,用的是常量池中的引用

```

字符串常见问题与练习

字符串比较

```
String s1 = new String("hello");
String s2 = new String("hello");
System.out.println(s1 == s2); // false
System.out.println(s1.equals(s2)); // true

String s3 = new String("hello");
String s4 = "hello";
System.out.println(s3 == s4); // false
System.out.println(s3.equals(s4)); // true

String s5 = "hello";
String s6 = "hello";
System.out.println(s5 == s6); // true
System.out.println(s5.equals(s6)); // true
```

- == ,对于基本数据类型而言,比较的是内容,对于引用数据类型而言,比较的是引用变量,即所指向的地址
- equals方法是Object的方法,默认是比较2个对象的地址,若要比较内容,应当重写父类方法

String中重写的equals方法

```
public boolean equals(Object anObject) {
    if (this == anObject) {
        return true;
    }
    if (anObject instanceof String) {
        String anotherString = (String)anObject;
        int n = value.length;
        if (n == anotherString.value.length) {
            char v1[] = value;
            char v2[] = anotherString.value;
            int i = 0;
            while (n-- != 0) {
                if (v1[i] != v2[i]) {
                    return false;
                }
                i++;
            }
            return true;
        }
    }
    return false;
}
```

当前对象是"abc" 参数对象是"aaa"

判断数组长度是否相同

逐位比较

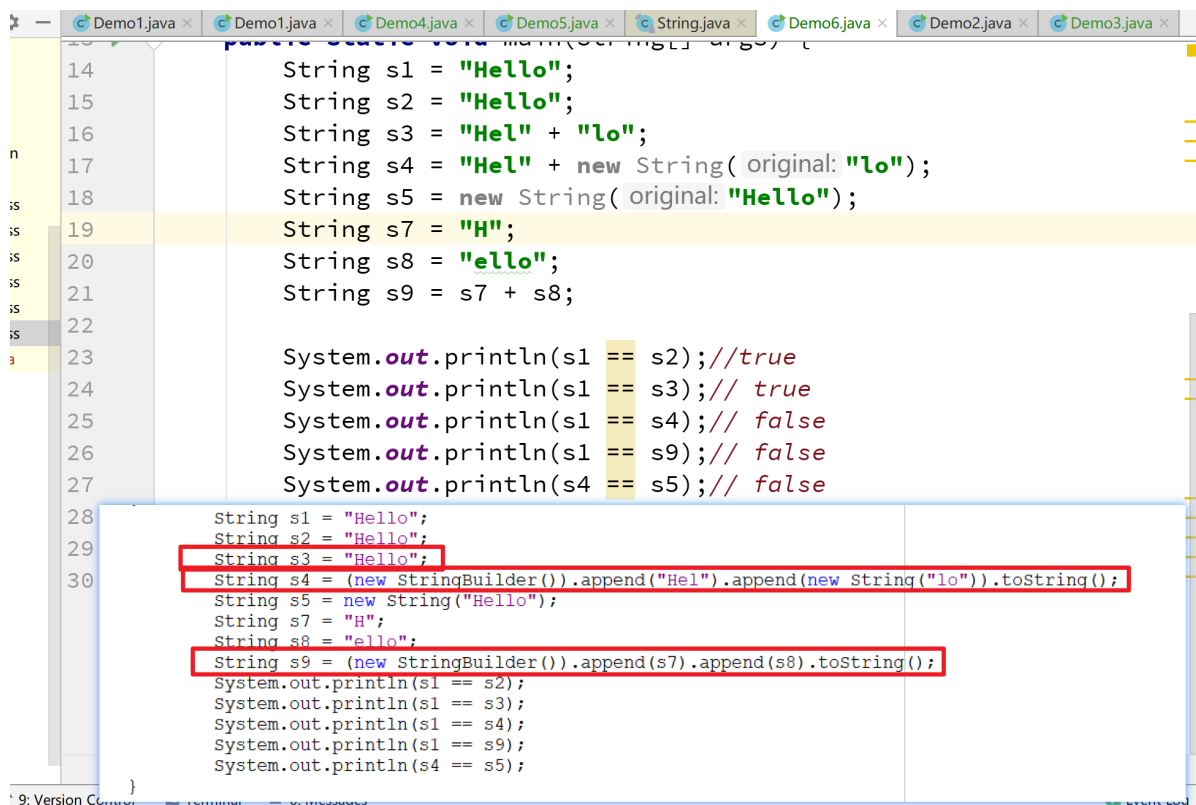
每位都相同 返回true

字符串拼接

```
String s1 = "Hello";
String s2 = "Hello";
String s3 = "Hel" + "lo";
String s4 = "Hel" + new String("lo");
String s5 = new String("Hello");
String s7 = "H";
String s8 = "ello";
String s9 = s7 + s8;

System.out.println(s1 == s2);
System.out.println(s1 == s3);
System.out.println(s1 == s4);
System.out.println(s1 == s9);
System.out.println(s4 == s5);
```

原理:



```
String s1 = "Hello";
String s2 = "Hello";
String s3 = "Hel" + "lo";
String s4 = "Hel" + new String("lo");
String s5 = new String("Hello");
String s7 = "H";
String s8 = "ello";
String s9 = s7 + s8;

System.out.println(s1 == s2); // true
System.out.println(s1 == s3); // true
System.out.println(s1 == s4); // false
System.out.println(s1 == s9); // false
System.out.println(s4 == s5); // false

String s1 = "Hello";
String s2 = "Hello";
String s3 = "Hello";
String s4 = (new StringBuilder()).append("Hel").append(new String("lo")).toString();
String s5 = new String("Hello");
String s7 = "H";
String s8 = "ello";
String s9 = (new StringBuilder()).append(s7).append(s8).toString();
System.out.println(s1 == s2);
System.out.println(s1 == s3);
System.out.println(s1 == s4);
System.out.println(s1 == s9);
System.out.println(s4 == s5);
```

进行字符串拼接的时候有2种情况

- 当参与字符串拼接的2个字符串,只要有1个引用变量的形式出现时,则会在堆上创建新的字符串对象.
 - 原因是因为参与了运算,无法在编译期确定其值,就不能在编译时期加入常量池
- 只有参与字符串拼接的2个字符串都是字面值常量的时候
 - 如果常量池中已有该字符串对象的引用,则返回常量池中的引用
 - 如果常量池中没有,则在堆上创建,并把引用放入常量池

String API

判断功能

用来比较字符串的内容，注意区分大小写

```
boolean equals(Object obj)
```

忽略字符串大小写比较字符串内容，常见用于比较网址URL

```
boolean equalsIgnoreCase(String str)
```

判断当前字符串对象是否包含，目标字符串的字符序列 "abc"

```
boolean contains(String str)
```

判断当前字符串对象，是否已目标字符串的字符序列开头

```
boolean startsWith(String str)
```

判断当前字符串，是否以目标字符串对象的字符序列结尾，常用于确定文件后缀名格式

```
boolean endsWith(String str)
```

判断一个字符串，是不是空字符串

```
boolean isEmpty()
```

获取功能

获取当前字符串对象中，包含的字符个数 "abcdef"

```
int length()
```

获取字符串对象代表字符序列中，指定位置的字符

```
char charAt(int index)
```

在当前字符串对象中查找指定的字符，如果找到就返回字符，首次出现的位置，如果没找到返回-1
也可以填字符

```
int indexOf(int ch)
```

指定从当前字符串对象的指定位置开始，查找首次出现的指定字符的位置，（如果没找到返回-1）
可以填入字符

```
int indexOf(int ch,int fromIndex)
```

查找当前字符串中，目标字符串首次出现的位置(如果包含)，找不到，返回-1
这里的位置是指目标字符串的第一个字符,在当前字符串对象中的位置

```
int indexOf(String str)
```

指定，从当前字符串对象的指定位置开始,查找首次出现的指定字符串的位置(如果没找到返回-1)
这里的位置是指目标字符串的第一个字符,在当前字符串对象中的位置

```
int indexOf(String str,int fromIndex) ,
```

返回字符串，该字符串只包含当前字符串中，从指定位置开始(包含指定位置字符)到结束的那部分字符串

```
String substring(int start)
```

返回字符串，只包含当前字符串中，从`start`位置开始(包含)，到`end`(不包含)指定的位置的字符串
`[start,end)`
`String substring(int start,int end)`

课堂练习：

1. 统计“abc”在字符串“abcdabcfgh”出现的次数
2. 借助于`int indexOf(String str,int fromIndex)`

```
package _15string.com.cskaoyan._03api;

/**
 * @description: 练习
 * @author: 景天
 * @date: 2022/7/21 9:59
 */
/**
课堂练习：
    1. 统计“abc”在字符串“abcdabcfgh”出现的次数
    2. 借助于int indexOf(String str,int fromIndex)

 */
public class Ex1 {
    public static void main(String[] args) {
        // 定义字符串
        String s = "abcdabcfgh";
        String s1 = "abc";
        // 定义计数器
        int count = 0;
        int fromIndex = 0;

        while ((fromIndex = s.indexOf(s1, fromIndex)) != -1) {
            // 统计
            count++;
            fromIndex++;
        }
        // 循环结束
        // 打印结果
        System.out.println("出现了"+count);
    }
}
```

转换功能

获取一个用来表示字符串对象字符序列的，字节数组

```
byte[] getBytes()
```

获取的是用来表示字符串对象字符序列的，字符数组

```
char[] toCharArray()
```

把字符数组转换成字符串

```
static String valueOf(char[] chs)
```

把各种基本数据类型和对象转换成字符串

```
static String valueOf(int i/double...)
```

把字符串全部转化为小写

```
String toLowerCase()
```

把字符串全部转换为大写

```
String toUpperCase()
```

字符串拼接，作用等价于 + 实现的字符串拼接

```
String concat(String str)
```

```
package _15string.com.cskaoyan._03api;
```

```
import org.junit.Test;
```

```
import java.util.Arrays;
```

```
/**
```

```
 * @description:
```

```
 * @author: 景天
```

```
 * @date: 2022/7/21 9:45
```

```
 **/
```

```
public class APITest {
```

```
    /*
```

```
    判断功能
```

```
    */
```

```
    @Test
```

```
    public void myTest1() {
```

```
        String s = "abcd";
```

```
        // 判断当前字符串对象，是否已目标字符串的字符序列开头
```

```
        //boolean startsWith(String str)
```

```
        System.out.println("s.startsWith(\"ab\") = " + s.startsWith("ab"));
```

```
        //判断当前字符串，是否以目标字符串对象的字符序列结尾，常用于确定文件后缀名格式
```

```
        //boolean endsWith(String str)
```

```
        System.out.println("s.endsWith(\"g\") = " + s.endsWith("g"));
```

```
        //判断一个字符串，是不是空字符串
```

```
        //boolean isEmpty() "" null
```

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

```

        System.out.println("\\".isEmpty() = " + "".isEmpty());

    }

    /*
    获取功能
    */
    @Test
    public void myTest2() {
        String s = "abcdefc";
        // 获取当前字符串对象中, 包含的字符个数 "abcdef"
        //int length()
        //
        //获取字符串对象代表字符序列中, 指定位置的字符
        //char charAt(int index)
        char c = s.charAt(0);
        System.out.println("c = " + c);

        //在当前字符串对象中查找指定的字符, 如果找到就返回字符, 首次出现的位置, 如果没找到返回-1
        //也可以填字符
        //int indexOf(int ch)
        int index = s.indexOf(99);
        System.out.println("index = " + index);

        //指定从当前字符串对象的指定位置开始, 查找首次出现的指定字符的位置, (如果没找到返回-1)
        //可以填入字符
        //int indexOf(int ch,int fromIndex)
        int index1 = s.indexOf(100, 1);
        System.out.println("index1 = " + index1);

        //查找当前字符串中, 目标字符串首次出现的位置(如果包含), 找不到, 返回-1
        //这里的位置是指目标字符串的第一个字符, 在当前字符串对象中的位置
        //int indexOf(String str)
        //
        //指定, 从当前字符串对象的指定位置开始, 查找首次出现的指定字符串的位置(如果没找到返回-1)
        //这里的位置是指目标字符串的第一个字符, 在当前字符串对象中的位置
        //int indexOf(String str,int fromIndex) ,
        //
        //返回字符串, 该字符串只包含当前字符串中, 从指定位置开始(包含指定位置字符)到结束的那部分字符串
        //String substring(int start)
        String substring = s.substring(1);
        System.out.println("substring = " + substring);

        //返回字符串, 只包含当前字符串中, 从start位置开始(包含), 到end(不包含)指定的位置的字符串[start,end)
        //String substring(int start,int end)
        String substring1 = s.substring(1, 3);
        System.out.println("substring1 = " + substring1);
    }

    /*
    转换功能
    */

```

```

@Test
public void myTest3() {
    String s = "abc";
    // 获取一个用来表示字符串对象字符序列的，字节数组
    //byte[] getBytes()
    byte[] bytes = s.getBytes();
    System.out.println(Arrays.toString(bytes));

    String s1 = new String(bytes);
    System.out.println(s1);
    //获取的是用来表示字符串对象字符序列的，字符数组
    //char[] toCharArray()
    char[] chars = s.toCharArray();
    System.out.println(Arrays.toString(chars));

    //把字符数组转换成字符串
    //static String valueOf(char[] chs)
    //
    //把各种基本数据类型和对象转换成字符串
    //static String valueOf(int i/double...)
    String s2 = String.valueOf(true);
    System.out.println("s2 = " + s2);

    //把字符串全部转化为小写
    //String toLowerCase()
    //
    //把字符串全部转换为大写
    //String toUpperCase()
    String s3 = s.toUpperCase();
    System.out.println("s3 = " + s3);
    //字符串拼接，作用等价于 + 实现的字符串拼接
    //String concat(String str)
    String s4 = s.concat("def");
    System.out.println("s4 = " + s4);
}
}

```

课堂练习：

1:字符串helloWORLD

2:第一个字符转为大写,其余字符转为小写 → Helloworld

```

package _15string.com.cskaoyan._03api;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/21 10:20
 */

/*

```

课堂练习:

1: 字符串 **helloWORLD**

2: 第一个字符转为大写, 其余字符转为小写 → **HelloWorld**

```
*/
public class Ex2 {
    public static void main(String[] args) {
        // 定义字符串
        String s = "helloWORLD";
        //func1(s);
        String str = s.substring(0,
1).toUpperCase().concat(s.substring(1).toLowerCase());
        System.out.println(str);

    }

    private static void func1(String s) {
        // 取出第一个字符
        String head = s.substring(0, 1);
        // 把第一个字符转大写
        String newHead = head.toUpperCase();
        // 截取剩余的
        String remind = s.substring(1);
        // 转成小写的
        String newRemind = remind.toLowerCase();
        // 重写拼接
        String newStr = newHead + newRemind;
        System.out.println(newStr);
    }
}
```

课堂练习:

1: 字符串反转

2: 举例:

键盘输入abc, 反转后结果为cba

```
package _15string.com.cskaoyan._03api;
```

```
import java.util.Scanner;
```

```
/**
 * @description:
 * @author: 景天
 * @date: 2022/7/21 10:28
 */
/**
```

课堂练习:

1: 字符串反转

2: 举例:

键盘输入abc, 反转后结果为cba

```

*/
public class Ex3 {
    public static void main(String[] args) {
        // 创建Scanner对象
        Scanner scanner = new Scanner(System.in);
        // 键盘接收数据
        String s = scanner.nextLine();
        // 把字符串转为char[]
        char[] chars = s.toCharArray();
        // 循环去处理,倒着遍历数组
        String str = "";
        // 定义空字符串 方便拼接
        for (int i = chars.length - 1; i >= 0; i--) {
            // 重写拼接
            str += chars[i];
        }
        // 输出结果
        System.out.println(str);
    }
}

```

其他功能

String类的替换功能

在新的字符串中,用新(new)字符,替换旧(old)字符"ab,cd"

`String replace(char old,char new)`

在新的字符串中,用新的字符串(new),替换旧(old)字符串

`String replace(String old, String new)`

在新的字符串中,去掉开头和结尾的空格字符

`String trim()`

分隔功能

将字符串按照符号分隔成字符串数组

`String[] split(String re)`

String类的比较功能

`int compareTo(String str)`

`int compareToIgnoreCase(String str)`

字符串的大小如何比较?

按照字典序,比较字符串的大小。字典序原本的含义实质,英文单词在字典中出现的先后顺序(在字典中,先出现的字符串小,后出现的字符串大).compareTo方法就是按照字典序进行比较的.

关于compareTo方法

1. 字符串长度一样,逐一比较返回第一个不一样字符的编码值的差值(调用者-参数)
2. 字符串长度不一样,并且前面的字符都相同,返回数组长度的差值(调用者-参数)
3. 长度一样,逐位字符也一样,返回0,表示相等

```

* Lexicographically greater than the string argument.
当前对象是"abc" 参数对象"abd"
public int compareTo( @NotNull 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;
}

```

课堂练习:

- 1:给出一句英文句子: "i want a bing dun dun"
- 2:每个单词的首字母都转换为大写并输出
- 3.使用split方法

```

package _15string.com.cskaoyan._03api;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/21 11:12
 */
/*
课堂练习:
1:给出一句英文句子: "i want a bing dun dun"
2:每个单词的首字母都转换为大写并输出
3.使用split方法

*/
public class Ex4 {
    public static void main(String[] args) {
        // 定义字符串
        String s = "i want a bing dun dun";
        // 进行分割 ---> String[]
        String[] strings = s.split(" ");
        // 遍历字符串数组
        // 空字符串 用来拼接
        String newStr = "";
        for (String str : strings) {
            // 首字母取出来 转换

```



```

        String substring = str.substring(0,
1).toUpperCase().concat(str.substring(1));
        // 拼接新的句子
        newStr+=substring+" ";
    }
    // 输出
    System.out.println(newStr.trim());
}
}

```

自然排序

课堂练习：

1:字符串bdcaegf

2:对字符串中的字符进行排序,最终得到结果 → abcdefg

```

package _15string.com.cskaoyan._04sort;

import java.util.Arrays;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/21 11:30
 */

/*
课堂练习：
    1:字符串bdcaegf
    2:对字符串中的字符进行排序,最终得到结果 → abcdefg
*/

public class Ex {
    public static void main(String[] args) {
        // 定义字符串
        String s = "bdcaegf";
        // 把字符串转字符数组char[]
        char[] chars = s.toCharArray();

        System.out.println("排序前:");
        System.out.println(Arrays.toString(chars));

        // 冒泡排序
        //bubbleSort(chars);

        // 简便方法
        Arrays.sort(chars);

        // 输出结果
        System.out.println("排序后:");
        System.out.println(Arrays.toString(chars));
    }
}

```

```

    }

    private static void bubbleSort(char[] chars) {
        for (int i = 0; i < chars.length - 1; i++) {
            // 两两交换 ,大的放后面
            for (int j = 0; j < chars.length - 1 - i; j++) {
                if (chars[j] > chars[j + 1]) {
                    char temp = chars[j];
                    chars[j] = chars[j + 1];
                    chars[j + 1] = temp;
                }
            }
        }
    }
}

```

Comparable接口

- 实现此接口的类，其对象数组（array）或对象容器（collection）
 - 就可以通过**Arrays.sort()**或**Collections.sort()**进行自动排序
- 对于实现该接口的A类来说，其对象a1.compareTo(a2)方法返回值
 - 小于0，表示a1对象小于a2，在自然排序中处于前面的位置
 - 大于0，表示a1对象大于a2，在自然排序中处于后面的位置
 - 等于0，表示a1对象等于a2

自定义类实现自然排序：

- 实现Comparable接口
- 重写compareTo方法

练习：

定义一个学生类，让其按照学生的年龄的大小，从小到大进行排序

```

package _15string.com.cskaoyan._04sort;

import java.util.Arrays;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/21 11:47
 */

public class Demo2 {
    public static void main(String[] args) {
        // 创建学生对象
        Student s1 = new Student("zs", 25, 66);
        Student s2 = new Student("ls", 22, 55);
        Student s3 = new Student("ww", 28, 88);
        Student s4 = new Student("zl", 20, 99);
    }
}

```

```

        Student s5 = new Student("长风", 25, 67);
        // 填充学生数组
        Student[] students = {s1, s2, s3, s4, s5};

        System.out.println("排序前:");
        System.out.println(Arrays.toString(students));
        // 排序
        Arrays.sort(students);
        // 输出结果
        System.out.println("排序后:");
        System.out.println(Arrays.toString(students));
    }
}

// 自定义类实现自然排序:
//实现Comparable接口
//重写compareTo方法
class Student implements Comparable<Student>{
    String name;
    int age;
    int score;

    public Student(String name, int age, int score) {
        this.name = name;
        this.age = age;
        this.score = score;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public int getScore() {
        return score;
    }

    @Override
    public String toString() {
        return "Student{" +
            "name='" + name + '\'' +
            ", age=" + age +
            ", score=" + score +
            '}';
    }

    @Override
    public int compareTo(Student o) {
        // 比较规则
        // 按照学生年龄从小到大进行排序
        // 按照学生年龄从大到小进行排序
    }
}

```

```

        //return this.getAge() - o.getAge();

        // 综合排序,年龄从小到大进行排序 如果年龄相同 按照分数从高到低
        //return o.getAge() - this.getAge();

        if (this.getAge() == o.getAge()) {
            return o.getScore() - this.getScore();
        }

        return this.getAge() - o.getAge();
    }

    //Override
    //public int compareTo(Object o) {
    //    return 0;
    //}
}

```

Comparator接口

在排序时需要注意一个比较特殊的方法，带比较器的Arrays.sort方法，

即sort(T[] a, Comparator<? super T> c)

根据指定比较器产生的顺序对指定对象数组进行排序。其中Comparator接口的实现类对象就是比较器，该对象通过compare方法传入比较的规则

表示传入比较规则的int compare(T o1, T o2)方法:

该方法可以看成是o1-o2,如果方法返回负数,o1<o2,相反则大于,只有当方法返回0时,才表示对象相等

三种方式去实现自然排序:

手写接口类实现

匿名内部类

lambda表达式

```

package _15string.com.cskaoyan._04sort;

import java.util.Arrays;
import java.util.Comparator;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/21 11:47
 */

public class Demo3 {
    public static void main(String[] args) {
        // 创建学生对象
        Student2 s1 = new Student2("zs", 25, 66);
        Student2 s2 = new Student2("ls", 22, 55);
        Student2 s3 = new Student2("ww", 28, 88);
        Student2 s4 = new Student2("zl", 20, 99);
        Student2 s5 = new Student2("长风", 25, 67);
    }
}

```

```

// 填充学生数组
Student2[] students = {s1, s2, s3, s4, s5};

System.out.println("排序前:");
System.out.println(Arrays.toString(students));
// 排序
//Arrays.sort(students, new MyComparator());
//匿名内部类
//Arrays.sort(students, new Comparator<Student2>() {
//    @Override
//    public int compare(Student2 o1, Student2 o2) {
//        // 按照分数从高到低
//        return o2.getScore()-o1.getScore();
//    }
//});
//lambda表达式
Arrays.sort(students, (stu1,stu2)->stu1.getScore()-stu2.getScore());

// 输出结果
System.out.println("排序后:");
System.out.println(Arrays.toString(students));

}
}

// 自定义类实现自然排序:
//实现Comparable接口
//重写compareTo方法
class Student2{
    String name;
    int age;
    int score;

    public Student2(String name, int age, int score) {
        this.name = name;
        this.age = age;
        this.score = score;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public int getScore() {
        return score;
    }

    @Override
    public String toString() {
        return "Student2{" +
            "name='" + name + '\'' +

```

```

        ", age=" + age +
        ", score=" + score +
        '}'';
    }

}

// 手写接口类实现
class MyComparator implements Comparator<Student2>{

    @Override
    public int compare(Student2 o1, Student2 o2) {
        // 按照年龄从小到大进行排序

        //return o1.getAge()-o2.getAge();
        return o2.getAge()-o1.getAge();
    }
}

```

可变长字符串

如果一个空字符串 "", 让其拼接10000次, 效率怎么样?

我们如果对字符串进行拼接操作, 每次拼接, 都会构建一个新的String对象, 既耗时, 又浪费空间。

StringBuffer构造方法

```

public StringBuffer() // 默认容量是16
public StringBuffer(int capacity) // 容量传多少就是多少
public StringBuffer(String str) // 容量是str的长度+16

```

StringBuffer成员方法

获取功能

`public int capacity()` 返回当前容量, 数组的长度, 理论值

`public int length()` 返回长度(字符的个数), 实际值

添加功能

`public StringBuffer append(String s)` 将指定的字符串(其他类型有重载方法)追加到此字符序列的尾部

在指定位置把任意类型的数据插入到字符串缓冲区里面

`public StringBuffer insert(int offset, String str)`

删除功能

`public StringBuffer deleteCharAt(int index)`: 删除指定位置的字符

`public StringBuffer delete(int start, int end)`: 删除从指定位置开始指定位置结束的内容

替换功能

使用给定String中的字符替换词序列的子字符串中的字符

```
public StringBuffer replace(int start,int end,String str)
```

反转功能

```
public StringBuffer reverse(): 将此字符序列用其反转形式取代, 返回对象本身
```

常见问题

String, StringBuffer之间的相互转换

```
package _15string.com.cskaoyan._05stringbuffer;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/21 15:15
 */
// 常见问题
public class Demo3 {
    public static void main(String[] args) {
        // String, StringBuffer之间的相互转换
        // String-----> StringBuffer
        String s = "abc";
        StringBuffer stringBuffer = new StringBuffer(s);
        //System.out.println(stringBuffer.equals(s));

        // StringBuffer----->String
        String s1 = stringBuffer.toString();
        System.out.println(s1);
    }
}
```

String, StringBuffer和StringBuilder有啥区别

StringBuffer和StringBuilder从效率上来说哪个更快?

- 和 String 类不同的是, StringBuffer 和 StringBuilder 类的对象能够被多次的修改
 - 并且不产生新的未使用对象, 不会产生效率问题和空间浪费问题
- StringBuffer是线程安全的, StringBuilder是线程不安全的
 - StringBuilder的效率会比StringBuffer效率更高, 单线程的程序推荐使用StringBuilder
 - 在多线程的程序中, 应该优先考虑使用StringBuffer, 安全性要更重要
 - 它们的效率都比String高很多

intern方法(了解)

intern方法是一个native方法，该方法首先会从字符串常量池中检测该对象的引用是否已存在(以Java8版本为标准)：

如果存在就返回字符串常量池中，该对象的引用

如果不存在就将存在于堆上的字符串对象的引用存入常量池（注意不会在常量池中创建新对象）

```
String s1 = "Hello";
String s2 = new String("Hello");
String s3 = new String("World");
String s4 = s2 + s3;
System.out.println(s1 == s2.intern());
System.out.println(s2 == s2.intern());
System.out.println(s3 == s3.intern());
System.out.println(s4 == s4.intern());
```

```
String str = new String("h1") + new String("h2");
String str2 = str.intern();
String str3 = "h1h2";
System.out.println(str == str2);
System.out.println(str == str3);
System.out.println(str2 == str3);
```