Java字符串

本章内容

- String类
- StringBuffer类
- main方法参数及+运算符
- 小结

1 String类

- 字符串是unicode字符的序列
- 字符串当作对象处理
- 字符串类型：String或StringBuffer
- String型字符串是不可更改的
- String类中的方法侧重于字符串的比较、字符定位、子串提取等查询操作。
- String类中的有些方法也会对字符串进行更改操作。但这些方法的调用都会产生一个新的字符串作为处理结果，而不会对原来字符串作任何修改。

String类接口：

```
public final class String
{     //构造方法
      public String();
      public String(String value);
      public String(char[] value);
      public String(byte[] bytes);
      public String(StringBuffer buffer);
      public String(byte[] bytes, String enc)
            throws UnsupportedEncodingException;
```

成员方法

```
//返回包含字符编码的字节数组
public byte[] getBytes();
public byte[] getBytes(String enc)
                    throws UnsupportedEncodingException;
//返回长度，即unicode字符的个数
public int length();
//提取index位置上的字符
public char charAt(int index);
//取子串
public String substring(int beginIndex, int endIndex);
```

```
//定位字符
public int indexOf(int ch);
//定位子串
public int indexOf(String str);
//大小比较(即差值大小)，相等(0)
public int compareTo(String anotherString);
//相等比较，内容比较，返回true/false
public boolean equals(Object anObject);
//字符串连接
public String concat(String str);
```

```
//替换字符
public String replace(char oldChar, char newChar);
//去掉前后空格
public String trim();
public String toLowerCase();
public String toUpperCase();
//返回字符串对象本身
public String toString();
//类方法:返回int(float,)型值的字符串表示
public static String valueOf(int i);      }
```

1.1 构造方法

类String中提供了下面的一些构造方法：

- 无参数的缺省的构造方法用来创建一个空串。

```
String s = new String();
```

- 利用已经存在的字符串常量创建一个新的String对象，该对象的内容与给出的字符串常量一致。

```
String s = new String("hello");
```

- 通过给构造方法传递一个字符数组可以创建一个非空串。

```
char chars[]= {'a' , 'b' , 'c' };
String s = new String( chars ) ;
```

```
Public class StringConstructors
{  Public static void main(String args[])
   {     String s,s1,s2,s3,s4,s5,s6,s7;
         byte byteArray[]={'J','a','v'};
         char charArray[]={'程','序','设','计'} ;
         StringBuffer sb=new StringBuffer("欢迎");
         s=new String("Hello!");
         s1=new String();
         s2=new String(s);
         s3=new String(sb);
         s4=new String(charArray,2,2);
         s5=new String(byteArray);
         s6=new String(charArray)
```

```
        System.out.println("s="+s);
        System.out.println("s1="+s1);
        System.out.println("s2="+s2);
        System.out.println("s3="+s3);
        System.out.println("s4="+s4);
        System.out.println("s5="+s5);
        System.out.println("s6="+s6);
    }
}
```

运行结果：

s=Hello!

s1=

s2=Hello!

s3=欢迎

s4=设计

s5=Jav

s6=程序设计

字符串的表示

- 字符串常量

字符串常量使用双引号括住的一串字符，比如：

"Hello world! "

Java编译器自动为每一个字符串常量生成一个String类的实例，因此可以用字符串常量直接初始化一个String对象，如：

```
String s="Hello world!";
```

1.2 提取与定位

- 提取:

  指从字符串中取得某个字符或者某个子串。

charAt(int index):

返回字符串中指定位置(index)上的字符(char型)。

substring(int beginIndex, int endIndex):

返回字符串中指定位置(从beginIndex至endIndex-1)上的子串(String型)。

- 定位:

指从字符串中搜索某个字符或者某个子串的位置。

indexOf(int ch)

返回指定字符(ch)在字符串中第一次出现的位置(int型)。

indexOf(String str)

返回指定子串(str)在字符串中第一次出现的位置(int型)。如果找不到指定的字符或子串，方法将返回-1。

lastIndexOf(int ch)与lastIndexOf(String str)

返回指定字符或子串在串中最后一次出现的位置

【例6-3】提取与定位举例

```
1) class Example0603 {
2)     public static void main(String[] args) {
3)         String s = "Java语言";
4)         int n1 = s.indexOf('a');
5)         int n2 = s.indexOf("a语");
6)         System.out.println("n1="+n1+"\n n2="+n2);
7)         char c = s.charAt(2);
8)         String s1 = s.substring(3,5);
9)         System.out.println("c = " + c + "\n  s1 = " + s1);
10)    }       }
```

输出结果:

```
n1 = 1
n2 = 3
c = v
s1 = a语
```

1.3 字符串比较

利用String类提供的有关方法可以对两个字符串进行大小比较，或者判断两个字符串是否相等。

compareTo(String anotherString)方法用于比较调用串与参数串之间的大小。

返回值=调用串－参数串

若调用串小于参数串，则返回值小于0；若调用串等于参数串(字符串长度及对应位置上的字符都相同)，则返回0；若调用串大于参数串，则返回值大于0。

```
1) class Example0604 {
2)     static void method(String[] arr) {
3)       for(int i = 0; i < arr.length-1; i++)
4)         for(int j = i + 1; j < arr.length; j++)
5)           if(arr[j].compareTo(arr[i]) < 0) {
6)             String t = arr[j];
7)             arr[j] = arr[i];
8)             arr[i] = t;
9)           }
10)    }
11)    public static void main(String args[]) {
12)      String[] strs = {"is", "the", "Now", "am", "Yes"};
13)      method(strs);
14)      for(int i = 0; i < strs.length; i++) {
15)            System.out.print(strs[i] + "  ");
16)      }
17)    }
18) }
```

【例6-4】编写method方法，其功能是对一个字符串数组进行排序。

输出结果是:
Now Yes am is the

注：

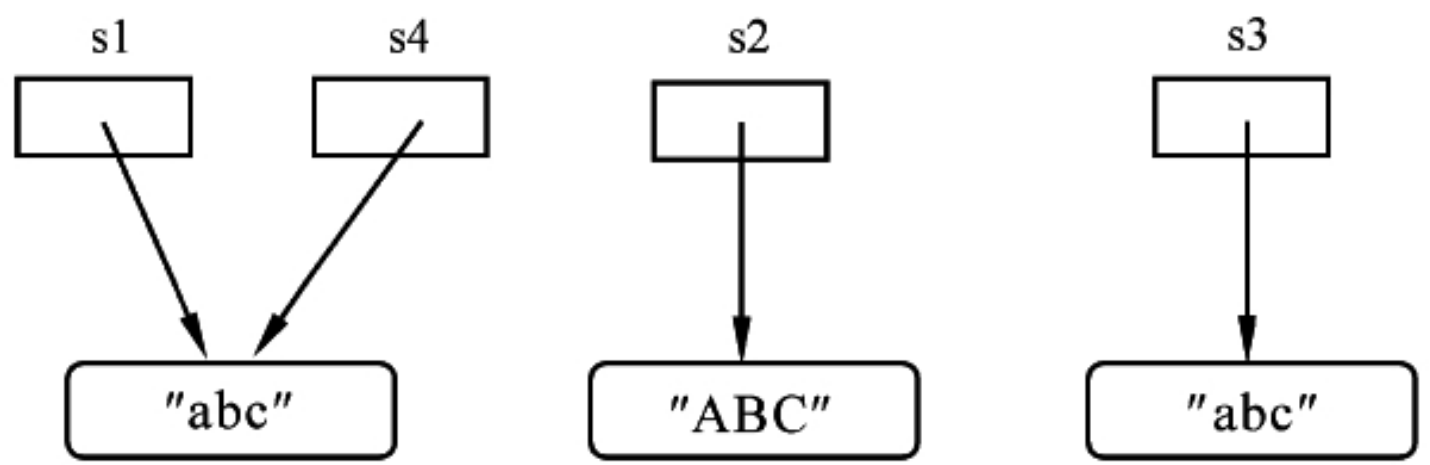
在Java中，两个字符串的大小比较最终会归结为两个字符串中各对应位置上字符的unicode码值的大小比较。从程序的输出结果可以看出，compareTo方法比较时区分字母的大小写。大写字母的unicode码值要比小写字母的unicode码值小。

String类的equals(Object anObject)方法用于判断两个字符串是否相等(内容相同)，其中参数对象必须是字符串对象。若调用串与参数串相等，则返回true；否则返回false。

另一个相似的方法是equalsIgnoreCase(String anotherString)。该方法在判断字符串相等时是否将忽略字符的大小写区别。

【例6-5】字符串相等性比较

```
1) class Example0605 {
2)     public static void main(String args[]) {
3)       String s1 = "abc";
4)       String s2 = "ABC";
5)       String s3 = new String(s1);
6)       String s4 = s1;
8)       System.out.println("s1 equals s2 : " + s1.equals(s2));
9)       System.out.println("s1 equalsIgnoreCase s2 : " +
                                    s1.equalsIgnoreCase(s2));
10)      System.out.println("s1 equals s3 : " + s1.equals(s3));
11)      System.out.println("s1 == s3 : " + (s1 = = s3));
12)      System.out.println("s1 == s4 : " + (s1 = = s4));
13)    }
14) }
```



例6-5
示意图

程序的输出结果如下:

s1 equals s2 : false

s1 equalsIgnoreCase s2 : true

s1 equals s3 : true

s1 == s3 : false

s1 == s4 : true

总结

运算符==与方法equals是两种不同的运算。

- ==用于判断两个引用类型变量是否指向同一个对象，即比较的是两个变量的引用值(地址)。
- String类的equals方法则用于判断两个字符串是否相等，即两个字符串是否具有完全相同的内容。
- compareTo方法也是对内容的比较。

1.4 其他实例方法

简单介绍其他几个实例方法的功能。这些方法的返回类型都为String型。

(1) 连接: concat(String str)

将参数串连接至调用串尾部产生一个新的字符串并返回。

例如:

"cares".concat("s") 返回 "caress"

"to".concat("get").concat("her") 返回 "together"

(2) 替换: replace(char oldChar, char newChar)

newChar字符替换字符串中出现的所有oldChar字符

注意:字符串对象是不可修改的。如果调用串中包含oldChar字符，那么方法将创建一个新的字符串对象。新的字符串与调用串相比，除了用newChar字符替换oldChar字符之外，其他部分是相同的。如果调用串中不包含oldChar字符，那么方法将返回调用串本身。例如:

```
String s1 = "abcDEFabc";
System.out.println(s1.replace('c','a')); //"abaDEFaba"
System.out.println(s1==s1.replace('x','y'));  //true
```

(3) 去空格: trim()。移去调用串前后两端的所有空白字符产生一个新的字符串并返回。如果调用串前后两端没有空白字符，或者调用串是空串，那么方法返回调用串本身。

(4) 大小写转换: toLowerCase()与toUpperCase()。

toLowerCase():将调用串中所有大写字母转换成小写字母产生一个新的字符串并返回。

toUpperCase() :将调用串中所有小写字母转换成大写字母产生一个新的字符串并返回。如果调用串中没有字符需要被转换，则返回调用串本身。例如:

"abcDEF".toLowerCase() 返回 "abcdef"

"abcDEF".toUpperCase() 返回 "ABCDEF"

1.5 类方法valueOf

String类
法的返回类
public static
这些valu
形式表示并
如果一个
valueOf(obj
另外：
Intege
Float.v

- **valueOf**

public static String valueOf(Object obj)

Returns the string representation of the Object argument.

Parameters:

obj - an Object.

Returns:

if the argument is null, then a string equal to "null"; otherwise, the value of obj.toString() is returned.

See Also:

Object.toString()

【例6-6】valueOf方法举例

```
class X
{   public String toString() //覆盖Object类中的toString()方法
    { return "example";   }
}
class Example0606
{  public static void main(String args[])
   {  char c = 0x41;
      int i = 0x41;
      boolean b= i==c;
      X obj=new X();
      char[] chars={'a','1','b','2'};
      System.out.print(String.valueOf(b) + "   ");
      System.out.print(String.valueOf(c) + "   ");
      System.out.print(String.valueOf(i) + "   ");
      System.out.print(String.valueOf(obj) + "   ");
      //同System.out.print(obj);
      System.out.println(String.valueOf(chars));     }
 }
```

输出结果为:

true A 65 example a1b2

2 StringBuffer类

- StringBuffer型字符串是可更改的。
- StringBuffer类中的方法侧重于字符的添加、插入、设置等更改操作。
- StringBuffer使用场合：对字符串频繁修改。

2.1 StringBuffer接口

```
public final class StringBuffer
{
   //构造方法
   //创建初始容量为16,不含任何字符(长度为0)的实例
   public StringBuffer();
   //创建初始容量为length,不含任何字符
   public StringBuffer(int length);
   //由str对象的内容创建一个StringBuffer实例，实例的长度为str.length()，初始容量为长度+16。
   public StringBuffer(String str);
```

```
//实例方法
//返回当前字符串包含的字符个数
public int length();
//返回StringBuffer类实例的当前容量
public int capacity();
//改变一个StringBuffer类实例的长度
public void setLength(int newLength);
//改变StringBuffer实例的容量
public void ensureCapacity(int minimumCapacity);
```

```
//在StringBuffer类型的字符串后追加字符串
public StringBuffer append(String str);
//插入字符串
public StringBuffer insert(int offset, String str);
//删除字符串
public StringBuffer delete(int start, int end);
//逆序
public StringBuffer reverse();
//由当前实例的内容创建一个String对象并返回
public String toString();   .....      }
```

2.2 基本方法

对StringBuffer类型字符串修改后的返回类型都为StringBuffer，且StringBuffer类中的这些方法都不会产生新的StringBuffer类实例，而仅仅是对原来实例内容的修改:

(1) setLength(int newLength)

● 若newLength小于原来的长度，则字符串尾部的一些字符将被剪裁;

● 若newLength大

加一些空字符('\u

```
StringBuffer str=new StringBuffer("12345");
str.setLength(10);
str.append("000");
System.out.println(str.length());   //13
System.out.println(str);            //12345
```

(2) ensureCapacity(int newCapacity)

为保证实例的容量(capacity)总是大于等于实例的长度(length)，在调整实例长度之前，方法首先会自动调用ensureCapacity方法以调整实例容量。

若newCapacity大于原来的容量，则实例的容量将被改变。新容量取下列值中较大者：

- 参数newCapacity的值；
- 原来容量乘2之后再加2。

若newCapacity小于原来的容量，则不处理。

例：

```
StringBuffer str=new StringBuffer(1);
System.out.println(str.length());     //length=0
System.out.println(str.capacity());  //capacity=1
str.append("000");                    //新插串长>capacity
System.out.println(str.length());     //length=3
System.out.println(str.capacity()); //capacity:2*1+2=4
str.append("1234");                   //length=7
System.out.println(str.capacity()); //?
str.ensureCapacity(8);                //8? or 10?
System.out.println(str.capacity()); //?
```

注：

一般情况下，程序员并不需要考虑容量的扩充，因为在用方法往**StringBuffer**类实例的中间或尾部插入或添加字符或字符串时，方法会自动调整实例的容量。

当然，如果预先能够确定一个**StringBuffer**类实例的长度会在处理中不断增加，那么也可以主动去设置一个比较大的容量，以避免频繁地扩充容量而带来的时间开销。

【例6-7】长度与容量比较举例

```
1) class Example0607 {
2)     public static void main(String args[]) {
3)       StringBuffer buf1=new StringBuffer("hello");//?
4)       StringBuffer buf2=new StringBuffer("he");    //?
5)       StringBuffer buf3=new StringBuffer("How do you do!");//?
6)       buf1.ensureCapacity(30);
7)       buf2.ensureCapacity(40);
8)       buf3.ensureCapacity(20);
9)       System.out.println("buf1="+buf1.length()+"  "
                                  + buf1.capacity());
10)      System.out.println("buf2="+buf2.length()+"  "
                                  + buf2.capacity());
11)      System.out.println("buf3="+buf3.length()+"  "
                                  + buf3.capacity());    }
12) }
```

```
buf1=5  44
buf2=2  40
buf3=14  30
```

(3) StringBuffer.append(Object obj)

将obj变为字符串添加至当前StringBuffer类实例内容的尾部。

- **StringBuffer append(boolean b)**
- **StringBuffer append(int i)**
- **StringBuffer append(long l)**

可将参数换为char、float、double、Object或者char[ ]型。

(4) insert(int offset, ...)

- **StringBuffer insert(int offset, String str)**

将str的内容插入至当前类实例的指定位置(offset)。offset的值必须大于等于0、小于等于当前StringBuffer类实例的长度。

- **StringBuffer insert(int offset, int i)**
- **StringBuffer insert(int offset, long l)**
- **StringBuffer insert(int offset, float f)**

第1个参数的类型都为int型，用于指定插入的位置；

第2个参数的类型除了可以是String型，还可以是char、boolean、double、Object或者char[]型。

(5) delete(int start, int end)。从StringBuffer类实例中删去指定位置(从start至end-1)上的一些字符。

"12345".delete(1,2);　　//1345

(6) replace(int start, int end, String str)。用str替换当前StringBuffer类实例指定位置(从start至end-1)上的一些字符。

```
StringBuffer buf1=new StringBuffer("12345");
System.out.println(buf1.replace(0,3,"b"));  //b45
```

(7) reverse()。将当前StringBuffer类实例中各字符的次序颠到一下。

"12345".reverse();　　//54321

【例6-8】append与insert方法举例

```
1) class Example0608 {
2)     public static void main(String args[]) {
3)       Object o="hello";
4)       String s="good bye";
5)       char charArray[]={'a','b','c','d','e','f'};
6)       boolean b=true;
7)       char c='A';
8)       int i=7;
9)       long l=10000000;
10)      float f=2.5f;
11)      double d=666.666;
12)
```

```
13)   StringBuffer buf=new StringBuffer();
14)   buf.insert(0,o).insert(0," ").insert(0,s);
15)   buf.insert(0," ").insert(0,charArray);
16)   buf.insert(0," ").insert(0,b);
17)   buf.append(" ").append(c).append(" ").append(i);
18)   buf.append(" ").append(l).append(" ").append(f);
19)   buf.append(" ").append(d);
20)   System.out.println(buf.toString());
21)   }
22) }
```

程序的输出结果为:

true abcdef good bye hello A 7 10000000 2.5 666.666

2.3 StringBuffer与StringBuilder

StringBuilder

- 由Java5.0引入，也表示一个可变的字符序列；
- 此类提供一个与 StringBuffer兼容的API，但不保证同步；
- 该类被设计用作StringBuffer的一个简易替换，用在字符串缓冲区被单个线程使用的时候（这种情况很普遍）；
- 如果可能，建议优先采用该类，因为在大多数实现中，它比 StringBuffer 要快。

3 main方法的参数及+运算符

3.1 main方法参数

public static void main(String[] argv)

Java解释器接受到下面的命令行

java [<选项>] <类文件> [<参数> ...]时，开始运行程序。

命令行中类文件名后的参数将传送给main方法的字符串数组参数args。参数的个数决定了字符串数组的大小:第1个参数存入args[0]，第2个参数存入args[1]，…。

程序运行时，命令行参数good被传送给argv[0]，参数morning被传送给argv[1]，所以程序的输出结果应该是D。

```
public class MyProg {
    public static void main(String[] argv) {
        System.out.println(argv[1])
    }
}
```

编译完成后，在命令行输入执行:

"java MyProg good morning"

请问输出的结果是什么？

A) java　B) myprog　C) good　D) morning

例：用符号#画一个宽为w高为h的空心方框，w和h用命令行参数提供。

```
//DoRect.java
public class DoRect
{  public static void main(String args[])
   {    int w=Integer.parseInt(args[0]);
        int h=Integer.parseInt(args[1]);
        Rectangle myrect=new Rectangle(w,h);
        myrect.drawRect();     }
}
//Rectangle.java
class Rectangle
{   int width,height,area;
    Rectangle(int w, int h)
    {     width=w;
          height=h;
    }
```

```
##############
#            #
#            #
##############
```

```
void drawRect()
{   int i,j;
    for(i=width;i>0;i--)
          System.out.print("#");
    System.out.println();
    for(i=height-2;i>0;i--){
            System.out.print("#");
            for(j=width-2;j>0;j--)
               System.out.print(" ");
           System.out.print("#");
            System.out.println();
     }
    for(i=width;i>0;i--)
          System.out.print("#");
    System.out.println("");
 }     }
```

3.2 运算符+

- 可以执行算术加操作，也可以执行字符串连接操作；
- 只要有一个操作数的类型为String型，该运算符就执行字符串连接操作；
- 为了避免产生过多的临时String对象，系统在执行字符串+操作时，往往会使用StringBuilder类以及相关的方法。比如表达式:

"x = " + 16 + y

实际上，JVM用下面表达式代替:

new StringBuilder("x=").append(16).append(y).toString()

可通过”javap –c 类名”来查看JVM结构上的描述。

运算符+是左结合的，并且在执行某个+运算(算术加或者字符串连接)时并不会考虑后面的+运算到底是算术加还是字符串连接。比如表达式:

"x = " + 10 + 20　　//"x = 1020"。

如果希望先将后面两个数值相加后再跟前面的字符串相连接，那么可以通过添加圆括号来改变其运算次序:

"x = " + (10 +20)

【例6-10】字符串连接运算举例

程序的输出如下：

x = null true1-2 12xy xy66 98

```
1) class Example0610
2) {   public static void main(String args[])
3)     {   Object o = null;
4)         System.out.print("x = " + o + "  ");
5)         System.out.print(true + "1-2" + "  ");
6)         System.out.print(6 + 6 + "xy" + "  ");
7)         System.out.print("xy" + 6 + 6 + "  ");
8)         System.out.println('a' + 1);
           //数字相加=char型+数字型
9)     }  }
```

4 小结

(1) 一个字符串是String类、StringBuffer 、StringBuilder类的一个实例。其中，String型字符串是不可更改的，而StringBuffer、StringBuilder型字符串是可更改的。

(2) String类中的有些实例方法的功能是对字符串进行更改操作，但对这些方法的调用并不会更改原来的字符串，而是产生一个新的字符串。

(3) String类的构造方法非常丰富: 利用字符数组创建String对象、利用字节数组创建String对象、利用已经存在的String对象创建新的String对象。

(4) String类提供有大量的实例方法，利用这些方法可:

- 从字符串内提取指定位置上的字符、子串(charAt、substring等)。
- 定位指定字符、子串在字符串内的位置(indexOf、lastIndexOf等)。
- 比较两个字符串的大小(compareTo、equals等)。
- 将两个字符串连接在一起(concat)。
- 对字符串内的字符进行替换(replace)。
- 去除字符串前后两端的空白符号(trim)。
- 转换字符串中字母的大小写(toLowerCase、toUpperCase)。

(5) String类包含一组重载的valueOf类方法，可以将各种类型的数据转换成字符串形式表示并返回。

(6) String类和StringBuffer类都有length()方法，可以返回各自实例的当前长度(即字符个数)。StringBuffer类还有capacity()方法，可以返回实例的当前容量。

(7) StringBuffer类有许多实例方法，利用这些方法可以对StringBuffer实例的内容进行追加(append)、插入(insert)、删除(delete)、颠倒(reverse)等操作。

(8) 字符串文字代表一个String对象，字符串文字是对该字符串对象的引用。

(9) 运算符+也可以执行字符串连接操作。如果另一个操作数不是String型，系统会在连接前自动将其转换成字符串。

下课！ 

Thank you!