第5章 Java中的常用类

学习目标

- 掌握 String 类和 StringBuffer 类的使用
- 了解 System 类和 Runtime 类中的常用方法
- 熟悉 Math 类和 Random 类的常用方法
- 掌握包装类和日期类的使用
- 掌握日期、时间格式器的使用

Java 提供了丰富的基础类库，通过这些类库可以提高开发效率，降低开发难度。因此，对于一个初学者来说，掌握 Java 基础类库中的一些常用类就显得十分重要。在基础类库中，提供了处理字符串、数学运算，以及日期和时间等功能的类，这些类是开发时常用的。本章将对基础类库中的这些常用类进行讲解。

5.1 String 类与 StringBuffer 类

在实际开发中会经常使用字符串，所谓的字符串就是指一连串的字符，它由许多单个字符连接而成。字符串中可以包含任意字符，这些字符必须包含在一对英文双引号("")之内，例如"abc"。Java 中定义了 String 和 StringBuffer 两个类来封装字符串，并提供了一系列操作字符串的方法。由于它们都位于 java.lang 包中，因此不需要导包就可以直接使用。本节将针对 String 类和 StringBuffer 类进行详细讲解。

5.1.1 String 类的初始化

在操作 String 类之前，首先需要对 String 类进行初始化。在 Java 中，可以通过以下两种方式对 String 类进行初始化，具体如下。

1. 使用字符串常量直接初始化一个 String 对象，其语法格式如下：

```
String 变量名=字符串;
```

在初始化字符串对象时，既可以将字符串对象的初始化值设为空，也可以初始化为一个具体的字符串，其示例如下：

```
String str1 =null;                    //初始化为空
String str2 ="";                      //初始化为空字符串
String str3 ="abc";                   //初始化为 abc,其中 abc 为字符串常量
```

2. 使用 String 的构造方法初始化字符串对象，其语法格式如下：

```
String 变量名 =new String(字符串);
```

在上述语法中，字符串同样可以为空或是一个具体的字符串。当为具体字符串时，会使用 String 类的不同参数类型的构造方法来初始化字符串对象。

String 类中包含多个构造方法，常用的构造方法如表 5-1 所示。

表 5-1　String 类的常用构造方法

方 法 声 明	功 能 描 述
String()	创建一个内容为空的字符串
String(String value)	根据指定的字符串内容创建对象
String(char[] value)	根据指定的字符数组创建对象

表 5-1 中列出了 String 类的 3 种构造方法，通过调用不同参数的构造方法便可完成 String 类的初始化。接下来通过一个案例来学习 String 类是如何通过构造方法来初始化字符串对象的，如例 5-1 所示。

例 5-1　Example01.java

```
1  public class Example01 {
2      public static void main(String[] args) {
3          //创建一个空的字符串
4          String str1 =new String();
5          //创建一个内容为 abc 的字符串
6          String str2 =new String("abc");
7          //创建一个内容为字符数组的字符串
8          char[] charArray =new char[] { 'A', 'B', 'C' };
9          String str3 =new String(charArray);
10         //输出结果
11         System.out.println("a" +str1 +"b");
12         System.out.println(str2);
13         System.out.println(str3);
14     }
15 }
```

运行结果如图 5-1 所示。

例 5-1 中，分别使用表 5-1 中的 3 个构造方法创建了字符串对象。其中第 4 行代码使用无参构造方法创建的是一个空字符串，所以第一个输出语句中的 str1 为空(" ")，当使用连字符＋连接 a 和 b 后，输出的结果为 ab。第 6 行代码使用参数类型为 String 的构造方法创建了一个内容为 abc 的字符串，第 8～9 行代码使用参数类型为字符数组的构造方法创建了一个内容为字符数组的字符串。从图 5-1 可以看出，它们最后的输出结果就是存储在字符

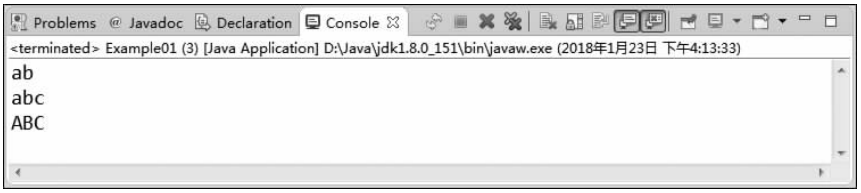

图 5-1 例 5-1 运行结果

串对象中的内容。

小提示:连接字符串可以通过运算符+来实现,例如上面案例代码("a" + str1 + "b")中的+的作用就是将两个字符串合并到一起并生成新的字符串。在 Java 中,如果+的两边操作数有一个为 String 类型,那么+就表示字符串连接运算符。

5.1.2 String 类的常见操作

String 类在实际开发中的应用非常广泛,因此灵活地使用 String 类是非常重要的,接下来讲解 String 类常用的一些方法,如表 5-2 所示。

表 5-2 String 类的常用方法

方法声明	功能描述
int indexOf(int ch)	返回指定字符在此字符串中第一次出现处的索引
int lastIndexOf(int ch)	返回指定字符在此字符串中最后一次出现处的索引
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
int lastIndexOf(String str)	返回指定子字符串在此字符串中最后一次出现处的索引
char charAt(int index)	返回字符串中 index 位置上的字符,其中 index 的取值范围是:0～(字符串长度－1)
boolean endsWith(String suffix)	判断此字符串是否以指定的字符串结尾
int length()	返回此字符串的长度
boolean equals(Object anObject)	将此字符串与指定的字符串比较
boolean isEmpty()	当且仅当字符串长度为 0 时返回 true
boolean startsWith(String prefix)	判断此字符串是否以指定的字符串开始
boolean contains(CharSequence cs)	判断此字符串中是否包含指定的字符序列
String toLowerCase()	使用默认语言环境的规则将 String 中的所有字符都转换为小写
String toUpperCase()	使用默认语言环境的规则将 String 中的所有字符都转换为大写
static String valueOf(int i)	返回 int 参数的字符串表示形式
char[] toCharArray()	将此字符串转换为一个字符数组
String replace(CharSequence oldstr, CharSequence newstr)	返回一个新的字符串,它是通过用 newstr 替换此字符串中出现的所有 oldstr 得到的
String[] split(String regex)	根据参数 regex(regex 是一个正则表达式,用来限定分隔规则)将字符串分割为若干个子字符串

续表

方 法 声 明	功 能 描 述
String substring(int beginIndex)	返回一个新字符串，它包含从指定的 beginIndex 起始角标处开始，直到此字符串末尾的所有字符
String substring(int beginIndex, int endIndex)	返回一个新字符串，它包含从指定的 beginIndex 起始角标处开始，直到索引 endIndex－1 角标处的所有字符
String trim()	返回一个新字符串，它去除了原字符串首尾的空格

在表 5-2 中列出了 String 类常用的方法，为了让读者更熟悉这些方法的作用，接下来通过几个案例来具体学习 String 类中常用方法的使用。

1. 字符串的基本操作

在程序中，需要对字符串进行一些基本操作，如获得字符串长度、获得指定位置的字符等。String 类针对每一个操作都提供了对应的方法，接下来通过一个案例来学习这些方法的使用，如例 5-2 所示。

例 5-2　Example02.java

```
1  public class Example02 {
2      public static void main(String[] args) {
3          String s ="abcabcbacdba";          //初始化字符串
4          System.out.println("字符串的长度为: " +s.length());
5          System.out.println("字符串中第一个字符:" +s.charAt(0));
6          System.out.println("字符 c 第一次出现的位置:" +s.indexOf('c'));
7          System.out.println("字符 c 最后一次出现的位置:" +s.lastIndexOf('c'));
8          System.out.println("子字符串第一次出现的位置: " +s.indexOf("ab"));
9          System.out.println("子字符串最后一次出现的位置: "
10                                                  +s.lastIndexOf("ab"));
11     }
12 }
```

运行结果如图 5-2 所示。

图 5-2　例 5-2 运行结果

从图 5-2 可以看出，String 类提供的方法可以很方便地获取字符串的长度，获取指定位置的字符以及指定字符和字符串的位置。

2. 字符串的转换操作

程序开发中,经常需要对字符串进行转换操作,例如将字符串转换成字符数组,将字符串中的字符进行大小写转换等。接下来通过一个案例来演示字符串的转换操作,如例 5-3 所示。

例 5-3 Example03.java

```
1  public class Example03 {
2      public static void main(String[] args) {
3          String str ="java";
4          char[] charArray =str.toCharArray();              //字符串转换为字符数组
5          System.out.print("将字符串转为字符数组的遍历结果:");
6          for (int i =0; i <charArray.length; i++) {
7              if (i !=charArray.length -1) {
8                  //如果不是数组的最后一个元素,在元素后面加逗号
9                  System.out.print(charArray[i] +",");
10             } else {
11                 //数组的最后一个元素后面不加逗号
12                 System.out.println(charArray[i]);
13             }
14         }
15         System.out.println("将 int 值转换为 String 类型之后的结果:"
16                             +String.valueOf(12));
17         System.out.println("将字符串转换成大写之后的结果:"
18                             +str.toUpperCase());
19     }
20 }
```

运行结果如图 5-3 所示。

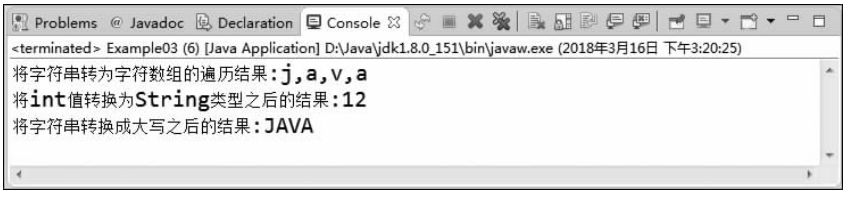

图 5-3 例 5-3 运行结果

例 5-3 中,使用 String 类的 toCharArray()方法将一个字符串转为一个字符数组,静态方法 valueOf()将一个 int 类型的整数转为字符串,toUpperCase()方法将字符串中的字符都转为大写。其中 valueOf()方法有很多重载的形式,float、double、char 等其他基本类型的数据都可以通过该方法转为 String 字符串类型。

3. 字符串的替换和去除空格操作

在开发程序的过程中,需要考虑到用户输入数据时会有一些错误和空格的情况,这时可以使用 String 类的 replace()和 trim()方法,进行字符串的替换和去除空格操作。接下来通过一个案例来学习这两个方法的使用,如例 5-4 所示。

例 5-4　Example04.java

```
1   public class Example04 {
2       public static void main(String[] args) {
3           String s ="     http :// localhost : 8080     ";
4           //字符串去除空格操作
5           System.out.println("去除字符串两端空格后的结果:" +s.trim());
6           //字符串替换操作
7           System.out.println("去除字符串中所有空格后的结果:"
8                              +s.replace(" ", ""));
9       }
10  }
```

运行结果如图 5-4 所示。

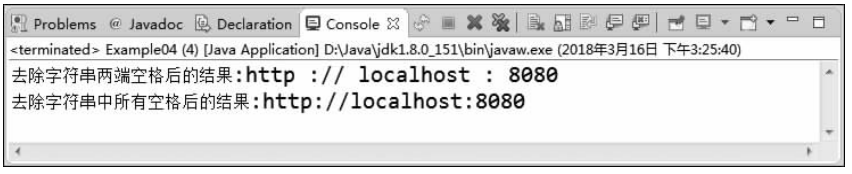

图 5-4　例 5-4 运行结果

例 5-4 中，调用了 String 类的两个方法，其中 trim()方法用于去除字符串中首尾的空格，replace()方法用于将字符串中所有与指定字符串匹配的子串替换成另一个字符串。

需要注意的是，trim()方法只能去除两端的空格，不能去除中间的空格。若想去除字符串中间的空格，则可以通过 String 类的 replace()方法来实现。

4. 字符串的判断操作

操作字符串时，经常需要对字符串进行一些判断，如判断字符串是否以指定的字符串开始、结束，是否包含指定的字符串，字符串是否为空等。在 String 类中针对字符串的判断操作提供了很多方法，接下来通过一个案例来学习这些判断方法的使用，如例 5-5 所示。

例 5-5　Example05.java

```
1   public class Example05 {
2       public static void main(String[] args) {
3           String s1 =" Starter";            //声明一个字符串
4           String s2 ="St";
5           System.out.println("判断是否以字符串 St 开头:" +s1.startsWith("St"));
6           System.out.println("判断是否以字符串 er 结尾:" +s1.endsWith("er"));
7           System.out.println("判断是否包含字符串 ar:" +s1.contains("ar"));
8           System.out.println("判断字符串是否为空:" +s1.isEmpty());
9           System.out.println("判断两个字符串是否相等:" +s1.equals(s2));
10      }
11  }
```

运行结果如图 5-5 所示。

在例 5-5 中涉及的方法都是用于判断字符串的，并且返回值均为 boolean 类型。在所使

图 5-5　例 5-5 运行结果

用的方法中,equals()方法是比较重要的,在 String 类中重写了父类 Object 中的 equals()方法。

在程序中可以通过==和 equals()两种方式对字符串进行比较,但这两种方式有明显的区别。equals()方法用于比较两个字符串中的字符值是否相等,==方法用于比较两个字符串对象的内存地址是否相同。对于两个字符串对象,当它们的字符值完全相同时,使用 equals 判断结果会是 true,但使用==判断时,结果一定为 false。为了便于理解,下面给出示例代码:

```
String str1 =new String("abc");
String str2 =new String("abc");
//结果为 false,因为 str1 和 str2 是两个对象
System.out.println(str1 ==str2);
//结果为 true,因为 str1 和 str2 字符内容相同
System.out.println(str1.equals(str2));
```

5. 字符串的截取和分割

在 String 类中针对字符串的截取和分割操作提供了两个方法,其中,substring()方法用于截取字符串的一部分,split()方法可以将字符串按照某个字符进行分割。接下来通过一个案例来学习这两个方法的使用,如例 5-6 所示。

例 5-6　Example06.java

```
1   public class Example06 {
2       public static void main(String[] args) {
3           String str ="2018-01-24";
4           //下面是字符串截取操作
5           System.out.println("从第 6 个字符截取到末尾的结果: "
6                              +str.substring(5));
7           System.out.println("从第 6 个字符截取到第 7 个字符的结果: "
8                              +str.substring(5, 7));
9           //下面是字符串分割操作
10          System.out.print("分割后的字符串数组中的元素依次为:");
11          //通过横线连接符"-"将字符串转换为字符串数组
12          String[] strArray =str.split("-");
13          //循环输出数组中的元素
14          for (int i =0; i <strArray.length; i++) {
15              if (i !=strArray.length -1) {
```

```
16                  //如果不是数组的最后一个元素,在元素后面加顿号
17                  System.out.print(strArray[i] +"、");
18              } else {
19                  //数组的最后一个元素后面不加顿号
20                  System.out.println(strArray[i]);
21              }
22          }
23      }
24  }
```

运行结果如图 5-6 所示。

图 5-6　例 5-6 运行结果

例 5-6 中，调用了 String 类中重载的两个 substring()方法，在第 6 行代码调用 substring(5)方法时，因为字符串中的字符索引是从 0 开始的，所以会截取字符串中第 6 个字符以及之后的所有字符。第 8 行代码调用 substring(5,7)方法时，会截取第 6 个和第 7 个字符。例程中的第 12～22 行代码演示了 split()方法的用法，该方法会根据指定的符号-将字符串分割成三部分，并存放到一个 String 类型的数组当中。使用 for 循环遍历数组即可按照要求输出所需内容，这里将各个日期之间使用顿号分隔。

脚下留心

String 字符串在获取某个字符时，会用到字符的索引，当访问字符串中的字符时，如果字符的索引不存在，则会发生 StringIndexOutOfBoundsException(字符串角标越界异常)。

接下来通过一个案例来演示这种异常，如例 5-7 所示。

例 5-7　Example07.java

```
1 public class Example07 {
2       public static void main(String[] args) {
3           String s ="abcde";
4           System.out.println(s.charAt(10));
5       }
6   }
```

运行结果如图 5-7 所示。

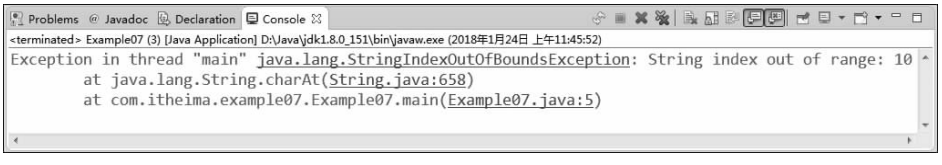

图 5-7　例 5-7 运行结果

从图 5-7 可以看出,访问字符串中的字符时,不能超出字符的索引范围,否则会出现异常,这与数组中的角标越界异常相似。

5.1.3 StringBuffer 类

在 Java 中,由于 String 类是 final 类型的,所以使用 String 定义的字符串是一个常量,因此它一旦创建,其内容和长度是不可改变的。如果需要对一个字符串进行修改,则只能创建新的字符串。为了便于对字符串进行修改,在 JDK 中提供了一个 StringBuffer 类(也称字符串缓冲区)来操作字符串。StringBuffer 类和 String 类最大的区别在于它的内容和长度都是可以改变的。StringBuffer 类似一个字符容器,当在其中添加或删除字符时,所操作的都是这个字符容器,因此并不会产生新的 StringBuffer 对象。

针对添加和删除字符的操作,StringBuffer 类提供了一系列的方法,如表 5-3 所示。

表 5-3 StringBuffer 类常用方法

方 法 声 明	功 能 描 述
StringBuffer append(char c)	添加字符到 StringBuffer 对象中末尾
StringBuffer insert(int offset,String str)	在 StringBuffer 对象中的 offset 位置插入字符串 str
StringBuffer deleteCharAt(int index)	移除 StringBuffer 对象中指定位置的字符
StringBuffer delete(int start,int end)	删除 StringBuffer 对象中指定范围的字符或字符串
StringBuffer replace(int start,int end,String s)	将 StringBuffer 对象中指定范围的字符或字符串用新的字符串 s 进行替换
void setCharAt(int index, char ch)	修改指定位置 index 处的字符
String toString()	返回 StringBuffer 缓冲区中的字符串对象
StringBuffer reverse()	将此 StringBuffer 对象用其反转形式取代

在表 5-3 中列出了 StringBuffer 类的一系列常用方法,对于初学者来说比较难以理解。接下来通过一个案例来学习一下表中方法的具体使用,如例 5-8 所示。

例 5-8 Example08.java

```
1  public class Example08 {
2      public static void main(String[] args) {
3          System.out.println("1. 添加-------------------------");
4          add();
5          System.out.println("2. 修改-------------------------");
6          update();
7          System.out.println("3. 删除-------------------------");
8          delete();
9      }
10     //添加
11     public static void add() {
12         StringBuffer sb =new StringBuffer();        //定义一个字符串缓冲区
13         sb.append("ABC");                           //添加字符串
14         System.out.println("append 添加结果: " +sb);
```

```
15          sb.insert(3, "DE");                    //在指定位置插入字符串
16          System.out.println("insert 添加结果: " +sb);
17      }
18      //修改
19      public static void update() {
20          StringBuffer sb =new StringBuffer("ABAAA");
21          sb.setCharAt(2, 'C');                  //修改指定位置字符
22          System.out.println("修改指定位置字符结果: " +sb);
23          sb.replace(3, 5, "DE");                //替换指定位置字符串或字符
24          System.out.println("替换指定位置字符(串)结果: " +sb);
25          System.out.println("字符串翻转结果: " +sb.reverse());
26      }
27      //删除
28      public static void delete() {
29          StringBuffer sb =new StringBuffer("ABCDEFG");
30          sb.delete(3, 7);                       //指定范围删除
31          System.out.println("删除指定位置结果: " +sb);
32          sb.deleteCharAt(2);                    //指定位置删除
33          System.out.println("删除指定位置结果: " +sb);
34          sb.delete(0, sb.length());             //清空缓冲区
35          System.out.println("清空缓冲区结果: " +sb);
36      }
37  }
```

运行结果如图 5-8 所示。

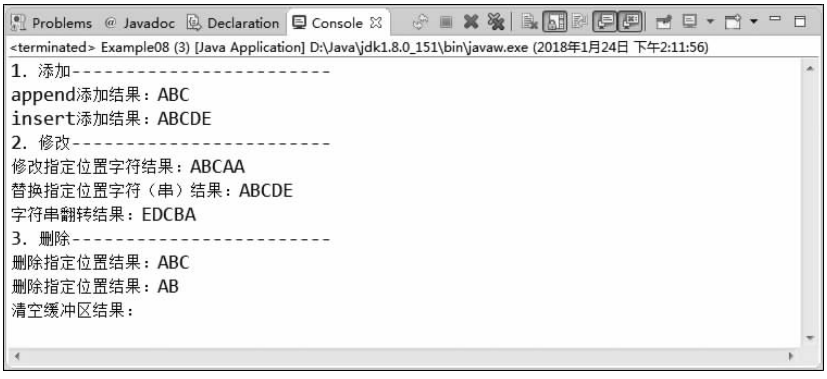

图 5-8　例 5-8 运行结果

在例 5-8 中涉及 StringBuffer 类的很多方法，其中 append()和 insert()方法是最常用的，并且这两个方法有很多重载形式，它们都用于添加字符。不同的是，append()方法始终将这些字符添加到缓冲区的末尾，而 insert()方法则可以在指定的位置添加字符。另外，StringBuffer 对象的 delete()方法用于删除指定位置的字符，包含起始索引，不包含结束索引，setCharAt()和 replace()方法用于替换指定位置的字符。

StringBuffer 类和 String 类有很多相似之处，初学者在使用时很容易混淆。接下来针对这两个类进行对比，简单归纳一下两者的不同，具体如下：

① String 类定义的字符串是常量，一旦创建后，内容和长度都是无法改变的。StringBuffer