第5章 chapter 5

常 用 类

教学重点	字符串操作常用类；包装类；日期类 Date 和格式化类 SimpleDateFormat；类 Calendar；类 Math；类 Random				
教学难点	类 StringTokenizer；常用类的主要方法的应用				
教学内容和教学目标	知 识 点	教学要求			
		了解	理解	掌握	熟练掌握
	类 String				√
	类 StringBuffer				√
	类 StringTokenizer			√	
	包装类				√
	日期类 Date				√
	格式化类 SimpleDateFormat				√
	类 Calendar			√	
	类 Math			√	
	类 Random			√	

Java 为编程者提供了功能强大的、大量的标准 API 包。学习 Java 不但要学会自己定义类，更重要的是在学习了 Java 基础编程知识后熟悉，掌握 Java 标准的 API，能够在不同的应用中使用它们。开发一个 Java 应用程序时，恰当地引用系统已定义的类可迅速构建应用程序，避免了从最低层的、不太熟悉的且烦琐的操作做起，在应用程序中可以直接引用，继承系统类，即实现了代码的重用性，避免了可能的错误，缩短了整个程序开发周期，又提高了编程效率。

本章主要介绍 Java 语言 API(应用程序接口)中的常用类，包括字符串、包装类、Date 类、Calendar 类、SimpleDateFormat 类、Math 类和 Random 类等。通过本章的学习使读者在程序中使用系统定义好的类来处理问题，提高编程效率。

5.1 字符串操作的常用类

字符串是内存中一个或多个连续排列的字符集合。在 Java 语言中,字符串将作为对象来处理,Java 提供的 java.lang 包中封装了字符串处理类 String、StringBuffer 和 StringTokenizer。

本节主要学习字符串操作的 3 个重要类:类 String、类 StringBuffer 和类 StringTokenizer。

5.1.1 类 String

Java 语言规定字符串常量必须用双引号""括起,一个字符串可以包含字母、数字和各种特殊字符,如+、-、*、/、$ 等。例如:

```
System.out.println("OK!");
```

Java 的任何字符串常量都是 String 类的对象,若没有命名,Java 自动为其创建一个匿名 String 类的对象,称为匿名 String 类的对象。

1. 创建 String 对象

类 String 提供了多种构造方法,它们可以用来在创建 String 对象时进行初始化。其中最常用的有两个。

(1) public String(String str):用一个已创建的字符串 str 创建另一个字符串。例如:

```
String s;
s=new String ("We are good friends!");
```

等价于

```
String s="We are good friends!" ;
```

(2) public String(char c[ ]):用字符数组 c 创建一个字符串对象。例如:

```
char[] c={'h','e','l','l','o'};
String str=new String(c);
char[] data={'壹','贰','叁','肆','伍'};  //Java 中一个汉字占一个字符
String str1=new String(data);
```

2. 类 String 的常用方法

类 String 提供了许多方法用于操作字符串。常用方法如下。

1) 求字符串长度

```
public int length()
```

例如：

```
String s=new String("我喜欢学 Java 语言");
int n=s.length();                              //n 的值是 10
```

注意：字符串中字符的位置(索引值)从 0 开始，最后一个字符的位置是 length()-1。

2) 查找单个字符或字符串

(1) public char charAt(int index)：返回当前串对象下标 index 处的字符。例如：

```
char ch=s.charAt(2);                           //ch='欢'
```

(2) public int indexOf(String s)：串从当前字符串头开始检索字符串 s，并返回首次出现 s 的索引位置。若找不到，则返回－1。例如：

```
int n="abcdefascd". IndexOf("cd");             //n 的值为 2
int m="abcd". IndexOf("Z");                    //m 的值为-1
```

(3) public int indexOf(String s, int start)：从当前下标 start 处检索，并返回首次出现 s 的索引位置。若找不到，则返回－1。例如：

```
String s="We are good students!";
int p=indexOf("st",2);                         //p 的值为 12
```

(4) public String substring(int begin)：返回当前串中从下标 begin 开始到串尾的子串。例如：

```
String s=new String("It is a good dog.");
String str=s.substring(6);                     //str 的值为"a good dog."
```

(5) public String substring(int begin, int end)：返回当前串中从下标 begin 开始到下标 end-1 结束的子串。例如：

```
String s="abcdefghijk".subString(2,5);         //s 的值为"cdef"
```

【例 5-1】 类 String 的常用方法应用举例。

```
//Example5_1.java
public class Example5_1{
    public static void main(String args[ ])  {
      String s1="Java Application";
      char cc[ ]={'J','a','v','a',' ','A','p','p','l','e','t'};
      String str=new String(cc);
      int len=str.length();
      int len1=s1.length( );
      int len2="ABCD".length( );
      char c1="12ABG".charAt(3);
      char c2=s1.charAt(3);

      int n1="abj".indexOf(97);
```

```
        int n2=s1.indexOf('J');
        int n3="abj".indexOf("bj",0);
        int n4=s1.indexOf("va",1);

        String s2="abcdefg".substring(4);
        String s3=s1.substring(4,9);
        System.out.println("s1="+s1+" len="+len1);
        System.out.println(str+" len="+len);
        System.out.println("ABCD=ABCD"+"  len="+len2);
        System.out.println("c1="+c1+"  c2="+c2);
        System.out.println("n1="+n1+"  n2="+n2);
        System.out.println("n3="+n3+"  n4="+n4);
        System.out.println("s2="+s2);
        System.out.println("s3="+s3);
    }
}
```

程序运行结果如图 5-1 所示。

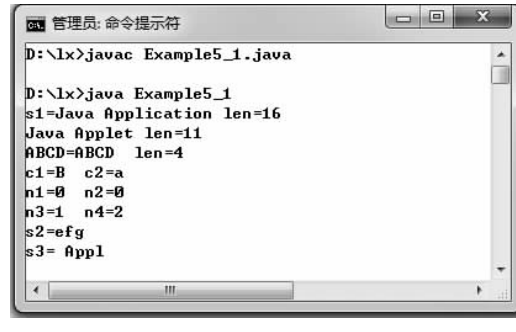

图 5-1　例 5-1 的运行结果

3）字符串比较的方法

(1) public int compareTo(String s)：按字典顺序与参数 s 指定的字符串比较大小。

① 当前字符串＝＝ s，返回值为 0。

② 当前字符串＞ s，返回值为正数。

③ 当前字符串 ＜ s，返回值为负数。

例如：

```
String s="uvwxyz";
int n=s.compareTo("girl");      //n 的值大于 0
```

(2) public boolean equals(String str)：比较当前字符串对象的实体是否与参数 str 指定字符串的实体相同，如果相同，返回值为 true，否则返回值为 false。例如：

```
String John=new String("笨鸟先飞");
String Jack=new String("天道酬勤");
```

```
String Rose=new String("笨鸟先飞");
boolean p,q;
p=John.equals(Jack);  //p=false
q=John.equals(Rose);  //q=true
```

(3) public boolean equalsIgnoreCase (String str)：在不区分字母的大小写时，比较当前字符串对象的实体是否与参数 str 指定字符串的实体相同，如果相同，返回值为 true，否则返回值为 false。

【例 5-2】 类 String 比较方法应用举例。

```
//Example5_2.java
public class Example5_2{
    public static void main(String args[ ]) {
      String s1="Java";
      String s2="java";
      String s3="Hello";
      String s4="Hello";
      String s5="Heloo";
      String s6="student";
      boolean b1=s1.equals(s2);
      boolean b2=s1.equals("world");
      boolean b3=s3.equals(s4);
      boolean b4=s1.equalsIgnoreCase(s2);
      int n1=s3.compareTo(s4);
      int n2=s1.compareTo(s2);
      int n3=s4.compareTo(s5);
      int d1=s6.compareTo("st");
      int d2=s6.compareTo("student");
      int d3=s6.compareTo("studentSt1");
      int d4=s6.compareTo("stutent");
      System.out.println("s1="+s1+"\ts2="+s2);
      System.out.println("s3="+s3+"\ts4="+s4);
      System.out.println("s5="+s5);
      System.out.println("equals: (s1==s2)="+b1+"\t(s1==world)="+b2);
      System.out.println("equals: (s3==s4)="+b3);
      System.out.println("equalsIgnoreCase: (s1==s2)="+b4);
      System.out.println("(s3==s4)="+n1+"\t(s1<s2)="+n2);
      System.out.println("(s4>s5)="+n3);
      System.out.println("d1="+d1+"\td2="+d2);
      System.out.println("d3="+d3+"\td4="+d4);
    }
}
```

程序运行结果如图 5-2 所示。

其余字符串操作方法请查阅 JDK 文档。

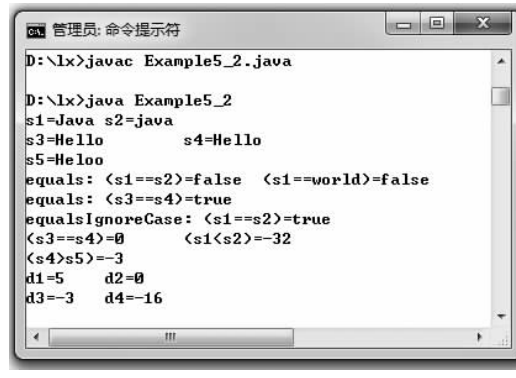

图 5-2 例 5-2 的运行结果

5.1.2 类 StringBuffer

类 StringBuffer 用于创建可变(可读/写)的字符串对象,而 String 类创建的字符串对象是不可修改的,即 String 对象一旦创建,实体就不可以再发生变化。

1. 类 StringBuffer 的构造方法

类 StringBuffer 提供了 3 种构造方法来创建一个 StringBuffer 对象。

1) public StringBuffer()

该方法无参数,创建了一个空的 StringBuffer 对象,此时对象的实体的初始内存空间为 16 个字符,当该对象的实体存放的字符序列的长度大于 16 时,实体的容量自动增加,以存放所增加的字符。例如:

```
StringBuffer  MyStr1=new StringBuffer();
```

2) public StringBuffer(int length)

该方法参数 length 指定对象的实体的初始内存空间容纳的字符个数,当实体存放的字符序列的长度大于 length 时,实体的容量自动增加,以存放所增加的字符。例如:

```
StringBuffer  MyStr2=new StringBuffer(20);
```

3) public StringBuffer(String str)

该方法参数 str 指定对象的实体中存放的字符串,其初始内存空间的大小为 str 的长度加上 16 个字符,当实体存放的字符序列的长度 length 时,实体的容量自动增加,以存放所增加的字符。例如:

```
StringBuffer  MyStr3=new StringBuffer("Hello everyone!");
```

2. 类 StringBuffer 的常用方法

1) public StringBuffer append(对象类型 参数对象名)

该方法的功能是将指定的参数对象转化成字符串，再追加到 StringBuffer 字符串对象中。参数可以是各种类型的对象，可以是 Object、String、char、字符数组、boolean、int、long、float、double。例如：

```
double d=22.22;
StringBuffer  MyStr1=new  StringBuffer();
MyStr1.append("你好!");
MyStr1.append(d);
System.out.println(MyStr1.toString());
```

输出结果：

```
你好!22.22
```

2) public StringBuffer insert(int n,对象类型 参数对象名)

该方法的功能是在指定的位置 n 插入参数对象，先将参数对象转化为字符串再插入。插入的参数对象可以是各种数据类型的数据。例如：

```
StringBuffer  MyStr2=new StringBuffer("Hello everyone!");
MyStr2.insert(3,423);
System.out.println(MyStr2.toString());
```

输出结果：

```
Hel423lo everyone!
```

需要注意的是，若希望将 StringBuffer 对象在屏幕上显示出来，则必须首先调用 toString()方法把它变成字符串常量。

3) public void setCharAt(int index, char ch)

该方法的功能是将指定位置 index 处的字符用给定的另一个字符 ch 来替换。例如：

```
StringBuffer  MyStr=new StringBuffer("Coat");
MyStr.setCharAt(0,'G');
System.out.println(MyStr.toString());
```

输出结果：

```
Goat
```

注意：当前对象实体中的字符串序列的第一个位置为 0，第二个位置为 1，依次类推。

4) public int length()

该方法的功能是获取字符串的长度。例如：

```
StringBuffer  MyStr=new StringBuffer("Coat");
int n=MyStr.length();
System.out.println(n);
```

输出结果：

```
4
```

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

该方法的功能是将当前 StringBuffer 对象实体中的字符序列的一个子字符序列用参数 str 指定的字符串替换,其中的一个子字符序列从下标 start 到 end-1 之间的字符。例如:

```
StringBuffer  Mybuff=new StringBuffer("欢迎学习 Java 语言!");
Mybuff.replace(4,8,"C");
System.out.println(Mybuff.toString());
```

输出结果:

```
欢迎学习 C 语言!
```

注意:4～7 之间的字符序列被替换。

6) public StringBuffer delete(int start,int end)

该方法的功能是将当前 StringBuffer 对象实体中的字符序列的一个子字符序列删除,其中的一个子字符序列从下标 start 到 end-1 之间的字符将被删除。例如:

```
StringBuffer  Mybuff1=new StringBuffer("欢迎学习高等数学!");
Mybuff1.delete(4,6);
System.out.println(Mybuff1.toString());
```

输出结果:

```
欢迎学习高等数学!
```

【例 5-3】 StringBuffer 类的使用。

```
//Example5_3.java
import java.io.*;
import java.lang.*;
public class Example5_3{
    public static void main(String args[ ]) {
      StringBuffer  MyStr=new StringBuffer("Coat");
      MyStr.setCharAt(0,'G');
      int n=MyStr.length();
      System.out.println(MyStr.toString());
      System.out.println(n);
      StringBuffer  Mybuff=new StringBuffer("欢迎学习 Java 语言!");
      Mybuff.replace(4,8,"C");
      System.out.println(Mybuff.toString());
      StringBuffer  Mybuff1=new StringBuffer("欢迎学习高等数学!");
      Mybuff1.delete(4,6);
      System.out.println(Mybuff1.toString());
```

```
    }
}
```

程序运行结果如图 5-3 所示。

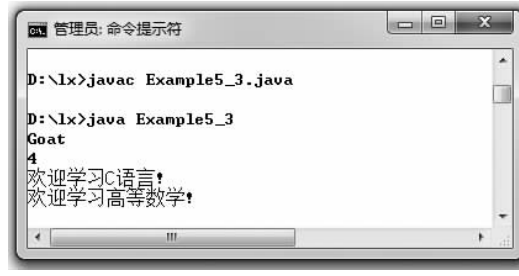

图 5-3 例 5-3 的运行结果

源程序说明：

(1) 导入 StringBuffer 所在的包 java.lang，对象实体中的字符串序列的第一个位置为 0，第二个位置为 1，依次类推。

(2) 熟练使用各方法，注意参数的取值范围。删除单个字符可以使用 StringBuffer 的 deleteCharAt(int index)方法；同样，替换单个字符可以使用 setCharAt()方法。

3. 字符串反转

```
public StringBuffer reverse()
```

该方法的功能是将当前 StringBuffer 对象实体中的字符序列按照相反的顺序进行排列。

【例 5-4】 字符串反转应用。

```
//Example5_4.java
import java.io.*;
import java.lang.*;
public class Example5_4
{
  public static void main (String args[ ])
  {
      StringBuffer  str=new StringBuffer("abcdefg");
        str.replace(3,6,"0101");
        str.append(345678);
        System.out.println(str.toString());
        str.reverse();
        System.out.println(str.toString());
  }
}
```

程序运行结果如图 5-4 所示。

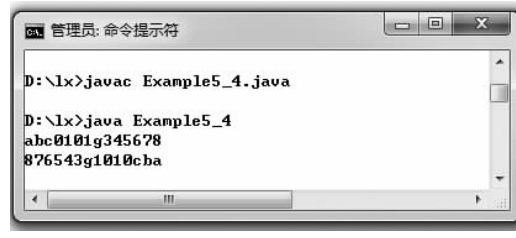

图 5-4　例 5-4 的运行结果

4. 字符串的加法和赋值

字符串是经常使用的数据类型，为了编程方便，Java 编译系统中引入了字符串的加法和赋值。参看下面的例子：

```
String  MyStr="Hello,";
MyStr=MyStr+"girls!";
```

这两个语句初看似乎有问题，因为 String 是不可变的字符串常量，实际上它们是合乎语法规定的，分别相当于：

```
String  MyStr=new StringBuffer().append("Hello,").toString();
MyStr=new StringBuffer().append(MyStr).append("girls!").toString( );
```

【例 5-5】 字符串加法和赋值。

```
//Example5_5.java
import java.io.*;
import java.lang.*;
public class Example5_5
{
  public static void main (String args[ ])
  {
    String  MyStr="Hello,";
    MyStr=MyStr+"girls!";
    System.out.println(MyStr.toString());
    String  MyStr1=new StringBuffer().append("Hello,").toString();
    MyStr1=new StringBuffer().append(MyStr1).append("girls!").toString( );
    System.out.println(MyStr1.toString());
  }
}
```

程序运行结果如图 5-5 所示。

由于这种赋值和加法的简便写法非常方便实用，所以在实际编程中用得很多。

5.1.3　类 StringTokenizer

当分析一个字符串并将字符串分解成可被独立使用的单词时，可以使用 java.util 包