第 3 章

数组与字符串

Chapter 3

实习学徒学习目标

(1) 掌握数组在内存中的分配状态。

(2) 掌握一维及二维数组的基本用法。

(3) 熟练使用 String 类型的常用方法。

数组是 Java 中一种重要的数据结构。数组是同类型数据的有序集合,同一数组里的每一个元素都具有一个相同的类型,先后顺序也是固定的。它的数据类型既可以是简单类型,也可以是类。对象数组和原始数据类型数组在使用方法上几乎完全一致。唯一的差别在于对象数组存储的是引用,而原始数据类型数组存储的是具体的数值。

声明一个数组是通过数组名进行的,而使用数组中存储的值时只能以数组元素为单位进行。一个数组中所拥有的元素数目称为该数组的长度。数组是一种高效的存储和随机访问对象引用序列的方式,使用数组可以快速访问数组中的元素。对于 Java 来说,为保存和访问一系列对象,最有效的方式就是数组。数组实际代表一个简单的线性序列,它使得元素的访问速度非常快,但我们也要为这种速度付出存储空间的代价。创建一个数组对象时,它的大小是固定的,而且不能在那个数组对象的“存储时间”内发生改变。

3.1 一 维 数 组

数组是一个独立的对象,数组在定义、分配内存和赋值后才可以使用。

3.1.1 数组的说明与构造

同其他类型变量一样,在使用数组前,必须先声明它。数组声明的格式为

```
类型 数组名[];
```

其中类型指出了数组中各元素的数据类型,它可以是基本类型和构造类型(类);数组名为 Java 标识符;[]部分指明该变量是一个数组类型变量。

例如:

```
int arry[ ];
```

说明是一个整数数组,数组中的每个元素为 int 整型数据。

数组声明时,也可以将[ ]放在类型之后;

例如:

```
int[ ] arry;
```

它的效果和上面的例子一样。

在说明数组时,不直接指出数组中元素的个数(即数组长度)。数组说明之后不能立即被访问,因为还没有为数组元素分配内存空间。需要使用 new 操作来构造数组,即在数组说明之后为数组元素分配内存空间,同时对数组元素进行初始化。其格式如下:

```
数组名=new 类型[数组长度];
```

例如:

```
arry=new int[5];
```

它为整型数组 arry 分配 5 个整数元素的内存空间,并使每个元素初值为 0。为简化起见,还可以把数组的说明和构造合并在一起,其格式如下:

```
类型 数组名[ ]=new 类型[数组长度];
```

例如:

```
int arry[ ]=new int[5];
```

这个语句等同于下面两个语句

```
int arry[ ];
arry=new int[5];
```

用 new 关键字为一个数组分配内存空间后,系统将为每个数组元素都赋予一个初值,这个初值取决于数组的类型。所有数值型数组元素的初值为 0,字符型数组元素的初值为一个不可见的 ISO 时控制符,布尔型数组元素的初值为 false,字符串数组和所有其他对象数组在构造该元素时的初始值为 null。在实际应用中,用户应根据具体情况对数组元素重新进行赋值。数组一旦创建之后,就不能再改变其长度。

3.1.2 数组的初始化

数组初始化就是为数组元素指定初始值。通常在构造数组时,Java 会使每个数组元素初始化为一个默认值。但在许多情况下,并不希望数组的初始值为默认值,此时,就需要用赋值语句来对数组进行初始化。

数组的初始化有两种方式:一种方式是像初始化简单类型一样自动初始化数组,即在说明数组的同时进行初始化;另一种方式是在声明之后再构造数组,然后为每个元素赋值。

例如:

```
int a[ ]={1,2,3,4,5};
```

上述语句声明并创建了数组 a,并且为数组的每个元素赋值,即初始化,使 a[0]=1,a[1]=2,a[2]=3,a[3]=4,a[4]=5。

由上可见,数组初始化可由花括号{ }括起来的一串由逗号分隔的表达式组成,逗号分隔数组元素中的值。在语句中不必明确指明数组的长度,因为已经体现在所给出的数组元素个数中了,系统会自动根据所给的元素个数为数组分配一定的内存空间,如上例中数组 a 的长度自动设置为 5。

3.1.3 数组元素的使用

声明了一个数组，并用 new 语句为它分配了内存空间后，就可以在程序中像使用任何变量一样来使用数组元素，即可以在任何允许使用变量的地方使用数组元素。数组元素的表示方式为

```
数组名[下标]
```

其中下标为非负的整数或表达式，其数据类型只能为 byte、short 和 int，而不能为 long。下标的取值范围从 0 开始，一直到数组的长度减 1。

Java 在对数组元素操作时会对数组下标进行越界检查，以保证安全性。若在 Java 程序中超出了对数组下标的使用范围，程序会有出错提示。在 Java 中，数组也是一种对象。数组经初始化后就确定了它的长度，对于每一个已分配了存储空间的数组，常用属性和方法表现在以下几个方面。

(1) 数组的复制：System. arraycopy( )。

(2) 数组的排序：Arrays. sort( )。

(3) 在已排序的数组中查找某个元素：Arrays. binarySearch( )。

(4) 数组中特定元素的寻找：binarySearch( )。

(5) 比较两个数组是否相等(在相同位置上的元素是否相等)：equals( )。

(6) 数组填充：fill( )。

(7) 数组排序：sort( )。

在 Java 中，所有的数组都有一个默认的属性 length，用于获取数组中元素的个数。例如 intArray. length 指明 intArray 的长度。

例 3.1 数组长度测定。

```
public class ArrayDemo1
{
  public static void main(String[] args)
  {
    // TODO Auto-generated method stub
    int i;
    double a1[];
    char [] a2;
    a1=new double[8];
    a2=new char[8];
    int a3[]=new int[8];
    byte[] a4=new byte[8];
    char a5[]={'A','B','C','D','E','F','H','I'};
    System.out.println("a1.length="+a1.length);
    System.out.println("a2.length="+a2.length);
    System.out.println("a3.length="+a3.length);
    System.out.println("a4.length="+a4.length);
    for(i=0;i<8;i++)
    {
      a1[i]=100.0+i;
      a2[i]=(char)(i+97);
      a3[i]=i;
    }
    System.out.println("a1\ta2\ta3\ta4\ta5");
    System.out.println("double\tchar\tint\tbtye\tchar");
```

```
    for(i=0;i<8;i++)
      System.out.println(a1[i]+"\t"+a2[i]+"\t"+a3[i]+"\t"+a4[i]+"\t"+a5[i]);
  }
}
```

程序执行结果如图 3.1 所示。

```
a1.length=8
a2.length=8
a3.length=8
a4.length=8
a1      a2      a3      a4      a5
double  char    int     btye    char
100.0   a       0       0       A
101.0   b       1       0       B
102.0   c       2       0       C
103.0   d       3       0       D
104.0   e       4       0       E
105.0   f       5       0       F
106.0   g       6       0       H
107.0   h       7       0       I
```

图 3.1　例 3.1 的运行结果

例 3.2　数组的循环遍历,求数组的最大值和最小值。

```
public class ArrayForDemo
{
  public static void main(String[] args)
  {
    // TODO Auto-generated method stub
    int [] score={88,78,87,96,60,93};
    int max;
    int sum;
    max=score[0];
    for(int i=1;i<score.length;i++)
    {
      if(score[i]>max)
      {
        max=score[i];
      }
    }
    System.out.println("最高成绩是:"+max);
    sum=0;
    for(int i=0;i<score.length;i++)
    {
      sum+=score[i];
    }
    System.out.println("平均成绩是:"+sum/score.length);
  }
}
```

程序执行的结果为

```
最高成绩是:96
平均成绩是:83
```

3.2 多维数组

Java 语言提供了支持多维数组的语法，但 Java 中只有一维数组，没有“多维数组”的明确的结构。然而对于一个一维数组而言，其数组元素可以是数组，这就是概念上的多维数组，也就是说，在 Java 语言中，把二维数组实际上看成每个数组元素是一个一维数组。这样的根本原因是计算机存储器的编址是一维的，即存储单元的编号是从 0 开始一直连续编到最后一个编号。在 Java 中，多维数组实际是数组的数组。定义多维数组变量要将每个维数放在它们各自的方括号中。

3.2.1 二维数组的声明

声明二维数组的方法有以下 3 种。这 3 种方法是等价的，一般使用第 3 种方法。

```
数据类型 数组名[ ][ ];
数据类型[ ] 数组名[ ];
数据类型[ ][ ] 数组名;
```

例如：

```
int [ ][ ] arry;
```

与一维数组一样，此时还没有为数组元素分配内存空间，还需要用 new 关键字来创建数组，然后才可以使用该数组的每个元素。

对二维数组来说，分配内存空间有下面两种方式。

(1) 直接为每一维数组分配空间，如：

```
int arry [ ][ ]=new int[2][3];
```

该语句创建了一个二维数组 arry，其较高一维数组含有两个元素，每个元素都是由三个整型数构成的整型数组。如图 3.2 所示。

arry[0][0]	arry[0][1]	arry[0][2]
arry[1][0]	arry[1][1]	arry[1][2]

图 3.2 二维数组 arry

(2) 从最高维开始，分别为每一维数组分配空间，如：

```
int score[ ][ ]=new int[ ][ ];          //最高维数组含 2 个元素，每个元素为一个整型数组
score[0]=new int[3];                    //最高维数组第一个元素是一个长度为 3 的整型数组
score[1]=new int[5];                    //最高维数组第二个元素是一个长度为 5 的整型数组
```

如图 3.3 所示。

score[0][0]	score[0][1]	score[0][2]		
score[1][0]	score[1][1]	score[1][2]	score[1][3]	score[1][4]

图 3.3 为数组分配空间

注意：在使用运算符 new 来分配内存时，对于多维数组至少要给出最高维数组的大小。如果程序中出现如下语句：

```
int score[ ][ ]=new int [ ][ ];
```

则编译出现问题。

3.2.2 二维数组的初始化

二维数组元素的初始化有以下两种。

(1) 直接对每个元素进行赋值。

(2) 在说明数组的同时进行初始化。

例如，如下语句：

```
int arry[ ][ ]={{1,4},{2,3},{6,7}};
```

声明了一个 2×3 的数组，并对每个元素赋值。即

```
arry[0][0]=1
arry[0][1]=4
arry[1][0]=2
arry[1][1]=3
arry[2][0]=6
arry[2][1]=7
```

3.2.3 二维数组的使用

对二维数组中的每个元素，其引用格式为

```
数组名[下标 1][下标 2];
```

其中，下标 1、下标 2 分别是第一维数组和第二维数组的下标。

例 3.3 使用二维数组，存放乘法表的结果。

```
public class ArrayDemo2
{
  public static void main(String[] args)
  {
    // TODO Auto-generated method stub
    int [][] triangleArray=new int[9][];
    for(int i=0;i<triangleArray.length;i++)
    {
      triangleArray[i]=new int[i+1];
    }
    for(int i=0;i<triangleArray.length;i++)
    {
      for(int j=0;j<triangleArray[i].length;j++)
      {
        triangleArray[i][j]=(i+1)*(j+1);
      }
    }
    for(int i=0;i<triangleArray.length;i++)
    {
      for(int j=0;j<triangleArray[i].length;j++)
```

```
        {
          System.out.print("\t"+triangleArray[i][j]);
        }
        System.out.println();
      }
    }
}
```

程序执行结果如图 3.4 所示。

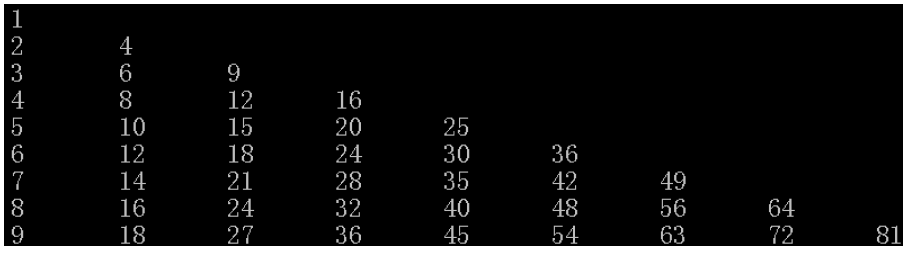

图 3.4 例 3.3 的运行结果

3.2.4 数组复制

在 Java 中,经常会用到数组的复制操作。一般来说,数组的复制是指将源数组的元素做副本,赋值到目标数组的对应位置。常用的数组复制方法有以下 3 种。

- 使用循环语句复制;
- 使用 clone()方法;
- 使用 System.arraycopy()方法。

1. 使用循环语句复制

使用循环语句访问数组,对其中每个元素进行访问操作,这是最容易理解、也是最常用的数组复制方式。例 3.4 中使用 for 循环实现数组复制功能。

例 3.4 使用 for 循环实现数组复制功能。

```
public class ArrayCopyFor
{
  public static void main(String[] args)
  {
    // TODO Auto-generated method stub
    int [] array1={1,2,3,4,5};
    int [] array2=new int[array1.length];
    for(int i=0;i<array1.length;i++)
    {
      array2[i]=array1[i];
    }
    System.out.println("复制结果:");
    for(int i=0;i<array2.length;i++)
    {
      System.out.print(array2[i]+",");
    }
  }
}
```

执行结果为

```
复制结果:
1,2,3,4,5,
```

2. 使用 clone()方法

在 Java 中,Object 类是所有类的父类,其中 clone()方法一般用于创建并返回此对象的一个副本,Java 中认为一切都是对象,所以使用该方法也可以实现数组的复制。

例 3.5 使用 clone()方法实现数组复制。

```
public class ArrayCopyClone
{
  public static void main(String[] args)
  {
    // TODO Auto-generated method stub
    int [] array1={1,2,3,4,5};
    int [] array2=array1.clone();
    System.out.println("复制结果:");
    for(int i=0;i<array2.length;i++)
    {
      System.out.print(array2[i]+",");
    }
  }
}
```

执行结果为

```
复制结果:
1,2,3,4,5,
```

3. 使用 System.arraycopy()方法

System.arraycopy()方法是 System 类的一个静态方法,可以方便地实现数组复制功能,System.arraycopy()方法的格式如下:

```
System.arraycopy(from,fromIndex,to,toIndex,count)
```

该方法共有 5 个参数:from、fromIndex、to、toIndex、count,其含义是将数组 from 中的索引为 fromIndex 开始的元素,复制到数组 to 中索引为 toIndex 的位置,总共复制的元素个数为 count 个。

例 3.6 使用 System.arraycopy()方法实现数组复制。

```
public class ArrayCopySystem
{
  public static void main(String[] args)
  {
    int [] array1={1,2,3,4,5};
    int [] array2=new int[array1.length];
    System.arraycopy(array1, 0, array2, 0, array1.length);
    System.out.println("复制结果:");
```

```
    for(int i=0;i<array2.length;i++)
    {
      System.out.print(array2[i]+",");
    }
  }
}
```

执行结果为

```
复制结果:
1,2,3,4,5,
```

3.2.5 数组应用实例

例 3.7 冒泡排序。

排序是把一组数据按照值的递增(由小到大)或递减(由大到小)的次序重新排列的过程。冒泡排序的关键点是从后向前对相邻的两个数组元素进行比较,若后面元素的值小于前面元素的值,则将这两个元素交换位置;否则不进行交换。依次进行下去,第一趟排序可将数组中值最小的元素移至下标为 0 的位置。对于有 n 个元素的数组,循环执行 $n-1$ 趟扫描便可完成排序。也可从前向后对相邻的两个数组元素进行比较,但此时应注意将大数向后移,与小数前移的冒泡法相对应。

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
class SortClass
{
  void sort(int arr[])
  {
    int i,j,temp;
    int len=arr.length;
    for(i=0;i<len-1;i++)
    for(j=len-1;j>i;j--)
    if(arr[j]<arr[j-1])
    {
      temp=arr[j-1];
      arr[j-1]=arr[j];
      arr[j]=temp;
    }
  }
}
public class Demo
{
  public static void main(String[] args) throws IOException
  {
    // TODO Auto-generated method stub
    BufferedReader keyin=new BufferedReader(new InputStreamReader(System.in));
    int i,j,temp;
    String c1;
    int arr[]=new int[5];
    int len=arr.length;
    System.out.println("请从键盘输入 5 个数据");
    for(i=0;i<len;i++)
```

```
    {
      c1=keyin.readLine();
      arr[i]=Integer.parseInt(c1);
    }
    System.out.print("原始数据为");
    for(i=0;i<len;i++)
    System.out.print(" "+arr[i]);
    System.out.println("\n");
    SortClass p1=new SortClass();
    p1.sort(arr);
    System.out.println("冒泡法排序的结果");
    for(i=0;i<len;i++)
      System.out.print(" "+arr[i]);
      System.out.println("\n");
  }
}
```

```
请从键盘输入5个数据
213
345
23
546
56
原始数据为 213 345 23 546 56

冒泡法排序的结果
 23 56 213 345 546
```

图 3.5　例 3.7 的运行结果

程序执行的结果如图 3.5 所示。

例 3.8　杨辉三角形。

杨辉三角形是宋朝数学家杨辉在公元 1261 年所著《详解九章算法》里面的一张图。

```
public class Demo
{
  public static void main(String[] args)
  {
    int triangle[][]=new int[10][];
    for (int i = 0; i < triangle.length; i++)
    {
      triangle[i]=new int[i+1];
      for(int j=0;j<=i;j++)
      {
        if(i==0||j==0||j==i)
        {
          triangle[i][j]=1;
        }else
        {
          triangle[i][j]=triangle[i-1][j]+triangle[i-1][j-1];
        }
        System.out.print(triangle[i][j]+"\t");
      }
      System.out.println();
    }
  }
}
```

执行结果如图 3.6 所示。

```
1
1       1
1       2       1
1       3       3       1
1       4       6       4       1
1       5       10      10      5       1
1       6       15      20      15      6       1
1       7       21      35      35      21      7       1
1       8       28      56      70      56      28      8       1
1       9       36      84      126     126     84      36      9       1
```

图 3.6　例 3.8 的运行结果