CHAPTER 第3章 数 组

本章学习重点：

- 了解 Java 数组的定义。
- 掌握 Java 数组的操作。
- 理解 Java 二维数组。
- 理解 Java 数组的引用传递。

在前面学习的整数类型、字符类型等都是基本数据类型，通过一个变量表示一个数据，这种变量被称为简单变量。但在实际中，经常需要处理具有相同性质的一批数据，这时可以使用 Java 中的数组，用一个变量表示一组性质相同的数据。数组是任何一种编程语言都不可缺少的数据类型。

3.1 一维数组

数组是一种数据结构，是按一定顺序排列的相同类型的元素集合。数组实际上就是一连串类型相同的变量，这些变量用一个名字命名，即数组名，并用索引区分它们。使用数组时，可以通过索引来访问数组元素，如数组元素的赋值和取值。

3.1.1 数组的声明

在 Java 中，数组是相同类型元素的集合，可以存放成千上万个数据。在一个数组中，数组元素的类型是唯一的，即一个数组中只能存储同一种数据类型的数据，而不能存储多种数据类型的数据。数组一旦定义好就不可以修改长度，因为数组在内存中所占大小是固定的，所以数组的长度不能改变，如果要修改就必须重新定义一个新数组或者引用其他的数组，因此数组的灵活性较差。

数组是可以保存一组数据的一种数据结构，它本身也会占用一个内存地址，因此数组是引用类型。

数组的声明包含两个部分：数组类型和数组的名称，定义数组的语法格式如下：

```
type[] 数组名
```

或

```
type 数组名[];
```

两种不同的语法格式声明的数组中，[ ]是一维数组的标识，它既可放置在数组名前面也可以放在数组名后面。例如：

```
int[] x;
float y[];
```

上述示例中声明了一个 int 类型的数组 x 与一个 float 类型的数组 y，数组名是用来统一这组相同数据类型的元素名称，数组中数组名的命名规则和变量相同。

3.1.2　数组的初始化

在 Java 程序开发中，使用数组之前都会对其进行初始化，这是因为数组是引用类型，声明数组只是声明一个引用类型的变量，并不是数组对象本身，只要让数组变量指向有效的数组对象，程序中就可以使用该数组变量来访问数组元素。所谓数组初始化，就是让数组名指向数组对象的过程，该过程主要分为两个步骤，一是对数组对象进行初始化，即为数组中的元素分配内存空间和赋值；二是对数组名进行初始化，即将数组名赋值为数组对象的引用。

通过两种方式可对数组进行初始化，即静态初始化和动态初始化，下面将演示这两种方式的具体语法。

1. 静态初始化

静态初始化是指由程序员在初始化数组时为数组每个元素赋值，由系统决定数组的长度。数组的静态初始化有两种方式，具体代码如下：

```
int[] x;
x=new int[]{10,20,30,40,50};
```

或

```
int x[]=new int[]{10,20,30,40,50}
```

对于数组的静态初始化也可以简写，具体代码如下：

```
int x[]={10,20,30,40,50}
```

上述示例中静态初始化了数组，其中大括号中包含数组元素，元素值之间用逗号“,”分隔。此处注意，只有在定义数组的同时执行数组初始化才支持使用简化的静态初始化。

2. 动态初始化

动态初始化是指由程序员在初始化数组时指定数组的长度，由系统为数组元素分配初始值，具体代码如下：

```
int a[]=new int[10];
```

上述示例会在数组声明的同时分配一块内存空间供该数组使用，其中数组长度是 10，由于每个元素都为 int 型数据类型，因此上例中数组占用的内存共 10×4＝40 字节。此外，动态初始化数组时，其元素会根据它的数据类型被设置为默认的初始值。常见的数据类型

默认值如表 3-1 所示。

表 3-1 数据类型默认值

数据类型	默认值	数据类型	默认值
byte	0	double	0.0D
short	0	char	空字符,'\n0000'
int	0	boolean	false
long	0L	引用数据类型	null
float	0.0F		

3.1.3 数组的访问

1. 访问数组

在 Java 中,数组对象有一个 length 属性,用于表示数组长度,所有类型的数组都是如此。获取数组长度的语法格式如下:

```
数组名.length
```

具体代码如下:

```
int[] list=new int[10];
int size=list.length;
```

数组中的变量又称为元素,每个元素都有下标(索引),下标从 0 开始,如在 int[] list = new int[10]中,list[0]是第 1 个元素,list[9]是第 10 个元素。因此,假如数组 list 有 n 个元素,那么 list[0]是第 1 个元素,而 list[n−1]则是最后一个元素。

如果下标值小于 0,或者大于或等于数组长度,编译程序不会报任何错误,但运行时出现异常。ArrayIndexOutOfBoundsException:N 为数组下标越界异常,N 表示试图访问的数组下标。

2. 数组遍历

数组的遍历是指依次访问数组中的每个元素。

【例 3-1】 数组遍历实例。

```
public class TestExample1 {
    public static void main(String[] args) {
        int[] i = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        for(int j = 0; j <i.length; j++) {
            System.out.print(i[j] +" ");
        }
        System.out.println();
        for(int j = i.length -1; j >= 0; j--) {
            System.out.print(i[j] +" ");
```

```
        }
    }
}
```

【例 3-2】　获取数组的最大值和最小值。

```
public class MaxMinDemo {
    public static void main(String[] args) {
        //定义数组
        int[] a = {83, 87, 24, 100, 68};
        int max = 0;
        int min = 0;
        max = min = a[0];
        for(int i=1; i<a.length; i++) {
            if(a[i] >max) {
                max = a[i];
            }
            if(a[i] <min) {
                min = a[i];
            }
        }
        System.out.println("最大值:"+max);
        System.out.println("最小值:"+min);
    }
}
```

3. 数组排序

数组排序是指数组元素按照特定的顺序排列。在实际应用中，经常需要对数据排序。数组排序有多种算法，本节介绍一种简单的排序算法——冒泡排序。这种算法是不断地比较相邻的两个元素，较小的向上冒，较大的向下沉，排序过程如同水中气泡上升，即两两比较相邻元素，反序则交换，直到没有反序的元素为止。

【例 3-3】　冒泡排序实例。

```
public class TestBubbleSort {
    public static void main(String[] args) {
        int[] array = {88, 62, 12, 100, 28};    //定义数组
        //外层循环控制排序轮数
        //最后一个元素,不用再比较
        for(int i=0; i <array.length-1; i++) {
            //内层循环控制元素两两比较的次数
            //每轮循环沉底一个元素,沉底元素不再参加比较
            for(int j = 0; j <array.length -1 -i; j++) {
                //比较相邻元素
                if(array[j] >array[j+1]) {
```

```
                    //交换元素
                    int tmp = array[j];
                    array[j] = array[j+1];
                    array[j+1] = tmp;
                }
            //System.out.print(array[j]+" ");
            }
            //打印每轮排序结果
            System.out.print("第"+(i+1)+"轮排序:");
            for(int j=0; j<array.length; j++) {
                System.out.print(array[j] +"\t");
            }
            System.out.println();
        }
        System.out.print("最终排序 :");
        for(int i=0; i<array.length; i++) {
            System.out.print(array[i] +"\t");
        }
        System.out.println();
    }
}
```

注意：可以用 Arrays.sort()方法对数组进行排序。

在例 3-3 中，外层循环是控制排序的轮数，每轮可以确定一个元素位置，由于最后一个元素不需要进行比较，因此外层循环的轮数为 array.length－1。内层循环控制每轮比较的次数，每轮循环沉底一个元素，沉底元素不用再参加比较，因此，内层循环的次数为 array.length－1－i。内层循环的次数被作为数组的索引，索引循环递增，实现相邻元素依次比较，如果当前元素大于后一个元素，则交换两个元素的位置。

3.1.4 数组的内存机制

数组是引用数据类型，因此数组变量就是一个引用变量，通常被存储在栈(Stack)内存中。数组初始化后，数组对象被存储在堆(Heap)内存中的连续内存空间，而数组变量存储了数组对象的首地址，指向堆内存中的数组对象。一维数组在内存中的存储原理如图 3-1 所示。

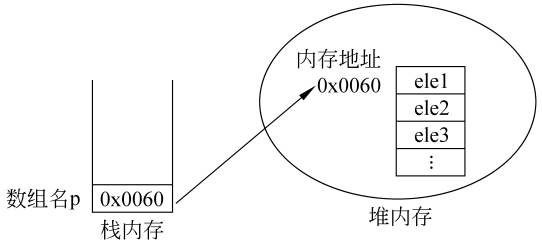

图 3-1 数组存储原理

在 Java 中，数组一旦初始化完成，数组元素的内存空间分配即结束，此后程序只能改变数组元素的值，而无法改变数组的长度。但程序可以改变一个数组变量所引用的数组，从而造成数组长度可变的假象。同理，在复制数组时，直接使用赋值语句不能实现数组的复制，这样做只是使两个数组引用变量指向同一个数组对象。

【例 3-4】 数组内存实例。

```
public class TestCopyArray {
    public static void main(String[] args) {
        int[] x = {88, 62, 12, 100, 28};
        //直接用赋值语句复制数组，赋的是数组的首地址
        int[] y = x;
        System.out.println(x);              //打印源数组名
        System.out.println(y);              //打印目的数组名
        x[0] = 22;                          //修改源数组
        System.out.println(y[0]);           //访问目的数组
    }
}
```

从程序运行结果可以发现，数组变量保存的就是数组首地址。通过赋值运算符复制数组，复制的是数组的首地址，源数组名和目的数组名都指向实际的数组内存单元，因此它们操作的是同一个数组，所以不能通过赋值运算符来复制数组，如图 3-2 所示。

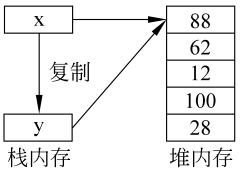

图 3-2　复制数组原理

【例 3-5】 复制数组实例。

```
public class TestExample1 {
    public static void main(String[] args) {
        int[] a = {1,2,3};
        int[] b = new int[a.length * 2];
        for(int i=0;i<a.length;i++){
            b[i] = a[i];
        }
        //int[] b = java.util.Arrays.copyOf(a, a.length * 2);
        a=b;
        print(a);
    }
    public static void print(int[] b){
        for(int i=0;i<b.length;i++){
            System.out.println("第"+i+"个元素的值为 "+b[i]);
        }
    }
}
```

3.2 二维数组

1. 二维数组的定义

二维数组可以看成以数组为元素的数组，常用来表示表格或矩形。二维数组的声明、初始化与一维数组类似。二维数组的声明代码如下：

```
int[][] a;
int a[][];
```

二维数组动态初始化代码如下：

```
a=new int[3][2]          //动态初始化 3×2 的二维数组
a[0]=new int[]{1,2}      //初始化二维数组的第一个元素
a[1]=new int[]{3,4}      //初始化二维数组的第二个元素
a[2]=new int[]{5,6}      //初始化二维数组的第三个元素
```

定义了一个 3 行 2 列的二维数组，即二维数组的长度为 3，每个二维数组的元素是一个长度为 2 的一维数组，二维数组元素的存储形式如图 3-3 所示。

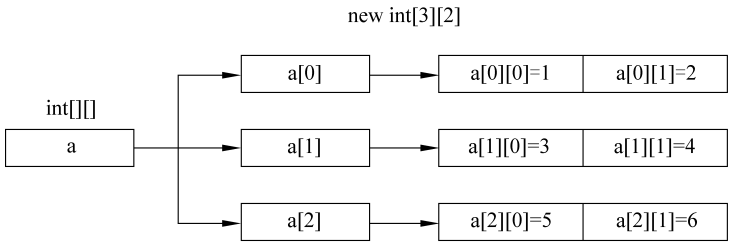

图 3-3 二维数组初始化

二维数组静态初始化语法格式：

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

或

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

二维数组的每个元素都是一个一维数组。二维数组 a 的长度是数组 a 的元素的个数，可由 a.length 得到；元素 a[i]是一个一维数组，其长度可由 a[i].length 得到。

【例 3-6】 二维数组实例。

```
public class MatrixAddition {
    public static void main(String args[]) {
        int i, j, k;
        //动态初始化一个二维数组
```

```
        int a[][] = new int[3][4];
        //静态初始化一个二维数组
        int b[][] = { { 1, 5, 2, 8 }, { 5, 9, 10, -3 }, { 2, 7, -5, -18 } };
        //动态初始化一个二维数组
        int c[][] = new int[3][4];
        for(i = 0; i <3; i++)
            for(j = 0; j <4; j++) {
                a[i][j] = (i +1) * (j +2);
            }
        for(i = 0; i <a.length; i++)
            for(j = 0; j <a[i].length; j++) {
                c[i][j] = a[i][j] +b[i][j];
            }
        //打印 Matrix C 标记
        System.out.println("*******Matrix C********");
        for(i = 0; i <c.length; i++) {
            for(j = 0; j <c[i].length; j++)
                System.out.print(c[i][j] +" ");
            System.out.println();
        }
    }

}
```

2. 锯齿数组

二维数组中的每行就是一个一维数组，因此，各行的长度就可以不同。这样的数组称为锯齿数组。创建锯齿数组时，可以只指定第一个下标，此时二维数组的每个元素为空，因此必须为每个元素创建一维数组。

【例 3-7】 锯齿数组实例。

```
public class TestJaggedArray{
    public static void main(String[] args) {
        //静态初始化锯齿数组
        int[][] array= {
                {1, 2, 3, 4, 5},
                {2, 3, 4, 5},
                {3, 4, 5},
                {4, 5},
                {5}
        };
        //动态初始化锯齿数组
        int[][] x = new int[5][];
        x[0] = new int[5];
        x[1] = new int[4];
```

```
            x[2] = new int[3];
            x[3] = new int[2];
            x[4] = new int[1];
            //为数组赋值
            for(int i = 0; i <x.length; i++) {
                for(int j = 0; j <x[i].length; j++) {
                    x[i][j] = array[i][j];
                }
            }
            //打印二维数组
            for(int i = 0; i <x.length; i++) {
                for(int j = 0; j <x[i].length; j++) {
                    System.out.print(x[i][j] +"\t");
                }
                System.out.println();
            }
        }
    }
```

首先,静态初始化锯齿数组 array 和动态初始化数组 x,然后通过嵌套 for 循环将锯齿数组 array 的数组元素值赋给锯齿数组 x 的数组元素,最后通过嵌套 for 循环将锯齿数组 x 的数组元素打印出来。

3.3 数组的引用传递

在 Java 中,可以使用数组作为方法的参数来传递数据。在使用数组参数时,应注意以下事项:

(1) 在形参列表中,数组名后的括号不能省略,括号的个数和数组的维数要相同,但在括号中可以不给出数组元素的个数。

(2) 在实参列表中,数组名后不需要括号。

(3) 数组名作为实参时,传递的是地址,而不是具体的数组元素值,即实参和形参具有相同的存储单元。

【例 3-8】 计算给定数组的平均值。

```
public class ArrayDemo {
    static float AverageArray(float a[]) {
        float average = 0;
        int i;
        for(i = 0; i <a.length; i++) {
            average = average +a[i];
        }
        return average / a.length;
```

```
    }
    public static void main(String[] args) {
        float average, a[] = { 1, 2, 3, 4, 5 };
        average = AverageArray(a);
        System.out.println("average=" +average);
    }
}
```

3.4　案例实现

【案例 3-1】随机点名

1. 案例描述

编写一个随机点名的程序，使其能够在全班同学中随机点中某一个同学的姓名。随机点名具有两个功能，一个是查看所有同学的姓名，一个是随机点取某一个同学的姓名。例如，班级一共有 5 名同学，先查看 5 名同学的姓名，然后随机选择一位同学输出名字。

2. 运行结果

案例运行结果如图 3-4 所示。

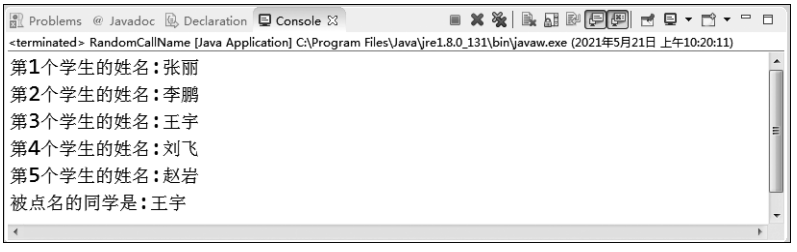

图 3-4　案例 3-1 运行结果

3. 案例思路

（1）在存储同学姓名时，如果对每个同学的姓名都定义一个变量进行存储，则会出现过多的变量，因此可以使用数组解决这个存储问题。

（2）通过对数组的遍历，可以打印出数组中的每个元素，即实现了对每个名字的预览。

（3）根据数组的长度，随机获取索引，通过索引获取数组中的元素，实现随机点名的功能。

（4）随机点名有两个功能，如果将所有代码都写在一起，则看起来非常冗长，可以针对不同功能的代码封装到不同的方法中，在 main()方法中调用即可。

4. 实现代码

案例的实现代码如下。

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

```
public class RandomCallName {
    public static void main(String[] args) {
        String[] names={"张丽","李鹏","王宇","刘飞","赵岩"};
        printNames(names);
        String name=randomName(names);
        System.out.println("被点名的同学是:"+name);
    }
    public static void printNames(String[] names) {
        for(int i=0;i<names.length;i++) {
            System.out.println("第"+(i+1)+"个学生的姓名:"+names[i]);
        }
    }
    public static String randomName(String[] names) {
        int index=new Random().nextInt(names.length);
        String name=names[index];
        return name;
    }
}
```

本 章 小 结

本章主要介绍 Java 语言中的数组部分，主要讲解了一维数组、二维数组及数组作为方法参数的用法。数组的使用过程分为声明、创建和访问。数组的声明只是对数组的定义过程，并不分配任何的存储空间，一定要在对数组初始化之后才可以访问数组中的数据元素。

习　　题

一、选择题

1. 获取数组 tmp 的长度用(　　)。

A. tmp.ArraySize;　　B. tmp.ArraySize();

C. tmp.length;　　D. tmp.length();

2. 若已定义 byte[ ] x= {11,22,33,-66};其中 0≤k≤3,则对 x 数组元素错误的引用是(　　)。

A. x[5-3]　　B. x[k]　　C. x[k+5]　　D. x[0]

3. 数组元素之所以相关，是因为它们具有相同的(　　)。

A. 空间　　B. 类型　　C. 下标　　D. 地址

4. 关于数组作为方法的参数时，向方法传递的是(　　)。

A. 数组的元素　　B. 数组的引用　　C. 数组的栈地址　　D. 数组自身

二、填空题

1. 一个数组中只能存储同一种_____的数据。

2. 数组的元素可以通过_____访问。

三、操作题

1. 验证 6174 问题(取任意一个 4 位数,4 个数字不能相同,将该数的 4 个数字重新组合,形成可能的最大数和可能的最小数,再将两者之间的差求出来;对此差值重复同样的过程,最后一定会得到 6174)。

2. 利用数组打印菱形。

3. 求前 100 个素数。

4. 用 * 打印如下的直角三角形。

```
*
**
***
****
```