


第 5 章

数组

( 视频讲解: 31 分钟)

数组是最为常见的一种数据结构，是相同类型的、用一个标识符封装到一起的基本类型数据序列或对象序列。可以用一个统一的数组名和下标来唯一确定数组中的元素。实质上数组是一个简单的线性序列，因此数组访问起来很快。本章将向读者介绍有关数组的知识。

通过阅读本章，您可以：

- » 掌握一维数组创建和使用的方法
- » 掌握多维数组创建和使用的方法
- » 了解如何遍历数组
- » 了解如何填充替换数组中的元素
- » 了解如何对数组进行排序
- » 了解如何复制数组



5.1 数组概述

 视频讲解：光盘\TM\lx\5\数组概述.exe

数组是具有相同数据类型的一组数据的集合。当需要使用的变量很多，而且数据类型相同时，逐个声明就显得非常麻烦，这时可以声明一个数组，然后对数组进行操作，从而省去了不少操作。例如，球类的集合——足球、篮球、羽毛球等；电器集合——电视机、洗衣机、电风扇等，就可以分别定义在一个数组中。Java 语言中虽然基本数据类型不是对象，但是由基本数据类型组成的数组则是对象，在程序设计中引入数组可以更有效地管理和处理数据。

数组根据维数的不同分为一维数组、二维数组和多维数组，大家习惯地将一维看成直线、二维看成平面、三维看成立体空间，那再多维又该怎么理解呢？其实 Java 语言中数组的维数并不用如此理解，这样非但不容易理解，反而会有使用的不便。下面给大家介绍一种简单的理解方法，如图 5.1 所示。

通过图示就能很好理解了，通俗地讲就是一维数组的每个基本单元都是基本数据类型的数据；二维数组就是每个基本单元是一维数组的一维数组；依此类推， n 维数组的每个基本单元都是 $n-1$ 维数组的 $n-1$ 维数组。下面通过三维数组实例进一步加深理解，如图 5.2 所示。

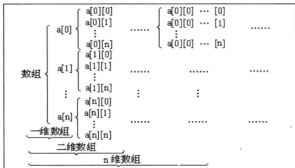


图 5.1 各维数组的理解图



图 5.2 三维数组示意图

图 5.2 中三维数组的元素都是一个二维数组，再把每个二维数组看成一个整体，则此三维数组也是一个二维数组的结构。

5.2 一维数组

 视频讲解：光盘\TM\lx\5\一维数组.exe

一维数组实质上是一组相同类型数据的集合，当需要在程序中处理一组数据或者传递一组数据时，可以应用这种类型的数组。本节将向读者介绍一维数组。

5.2.1 创建一维数组

数组作为对象允许使用 `new` 关键字进行内存分配。在使用数组之前，必须首先定义数组变量所属的类型，即声明数组。声明一维数组有两种形式，语法格式分别如下：

```
数组元素类型 数组名字[];  
数组元素类型[] 数组名字;
```

- ☑ 数组元素类型：决定了数组的数据类型，它可以是 Java 中任意的数据类型，包括基本数据类型和非基本数据类型。
- ☑ 数组名字：为一个合法的标识符。
- ☑ 符号“[]”：指明该变量是一个数组类型变量，单个“[]”表示要创建的数组是一维数组。

【例 5.1】 声明一维数组。

```
int arr[];           //声明 int 型数组，数组中的每个元素都是 int 型数值  
String []str;        //声明 String 数组，数组中的每个元素都是 String 型数值
```

声明数组后，还不能访问它的任何元素，因为声明数组仅仅是给出了数组名字和元素的数据类型，要想真正使用数组还要为其分配内存空间，且分配内存空间时必须指明数组的长度。分配内存空间的语法格式如下：

```
数组名字 = new 数组元素类型[数组元素的个数];
```

- ☑ 数组名字：已经声明的数组变量的名称。
- ☑ `new`：对数组分配空间的关键字。
- ☑ 数组元素的个数：指定数组中变量的个数。即数组的长度。

【例 5.2】 为数组分配内存。

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

上述代码表示为已经创建好的数组 `arr` 分配长度为 5 的内存空间，即该数组可以有 5 个元素。应该注意的是，在内存空间中数组的下标是从 0 开始的。`arr` 的内存存储状态如图 5.3 所示。

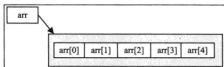


图 5.3 一维数组的内存模式



说明

使用 `new` 关键字为数组分配内存时，数组中各个元素的初始化值都为 0。

创建数组和分配内存不一定要分开执行，可以在创建数组时直接为变量赋值。语法格式如下：

数组元素类型 数组名[] = new 数组元素类型[数组元素的个数];

【例 5.3】 声明一维数组并分配内存。

```
int month[] = new int[12];
```

上述代码创建了一个一维数组 month，并指定了数组长度是 12。这种创建数组的方法也是 Java 程序编写过程中普遍的做法。



注意

无论用何种方法声明数组，前面的中括号中都不可以填写内容，否则将会在编译时出现错误。

5.2.2 初始化一维数组

数组可以与基本数据类型一样进行初始化操作，数组的初始化可分别初始化数组中每个元素。数组的初始化有两种形式。

【例 5.4】 两种方法初始化一维数组。

```
int arr[] = new int[]{1,2,3,5,25};           //第一种初始化方式
int arr2[] = {34,23,12,6};                   //第二种初始化方式
```

数组的初始化方式是：把数据类型包括在大括号之内，中间用逗号分开数组元素的值，系统自动为数组分配一定的空间。第一种初始化方式，创建 5 个元素的数组，其值依次为 1、2、3、5、25；第二种初始化方式，创建 4 个元素的数组，其值依次为 34、23、12、6。



说明

初始化数组时可以省略 new 运算符和数组的长度，编译器将根据初始值的数量来自动计算数组长度，并创建数组。

5.2.3 范例 1：求一维数组各元素的和

一维数组每个元素都有自己的值，使用 for 循环根据数组的下标，将数组的每个元素的值相加求和。运行结果如图 5.4 所示。（实例位置：光盘\TM\sl\5\1）

在项目中创建 SumNum 类，在类的主方法中使用数组存储 1~10 这 10 个整数，然后求出这 10 个数的和，并在控制台输出。代码如下：

```
public class SumNum {
    public static void main(String[] args) {
        int[] num = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };           //创建并初始化一维数组 num
```

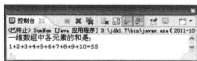


图 5.4 输出一维数组各元素的和

```

int sum = 0;                                //进行累加求和的变量 sum
System.out.println("一维数组中各元素的和是: ");
for (int i = 0; i < 10; i++) {               //通过 for 循环遍历数组
    if (i == 9) {                             //判断数组的下标是否为 9
        System.out.print(num[i] + "=");      //数组元素的下标是 9, 输出等号
    } else {
        System.out.print(num[i] + "+");      //数组元素的下标不是 9, 输出加号
    }
    sum = sum + num[i];                       //进行累加求和
}
System.out.println(sum);                     //输出和
}

```

5.2.4 范例 2: 获取一维数组的最小值

一维数组的每个元素可以不同, 有大有小, 通过 for 循环和 if 条件语句, 再根据数组下标求出数组元素中的最小值。运行结果如图 5.5 所示。(实例位置: 光盘\TM\sl\5\2)

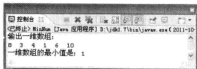


图 5.5 获取一维数组的最小值

在项目中创建 MinNum 类, 在类的主方法中创建一维数组, 求数组中值最小的元素, 并在控制台输出。代码如下:

```

public class MinNum {
    public static void main(String[] args) {
        int[] num = { 8, 3, 4, 1, 6, 10 };      //创建并初始化一维数组 num
        System.out.println("输出的一维数组: ");
        for (int i = 0; i < num.length; i++) {
            System.out.print(num[i] + " ");      //输出一维数组
        }
        int min = num[0];
        for (int j = 0; j < num.length - 1; j++) { //通过 for 循环遍历数组
            if (min > num[j + 1]) {
                min = num[j + 1];
            }
        }
        System.out.println("\n 一维数组的最小值是: " + min); //输出数组的最小值
    }
}

```

5.3 多维数组

 视频讲解：光盘\TM\lx\5\多维数组.exe

二维和多于二维的数组被统称为多维数组，其中二维数组在程序中经常会使用，三维数组也偶尔会使用，所以本节主要介绍二维数组和三维数组。

5.3.1 二维数组

如果一维数组中的各个元素仍然是一维数组，那么它就是一个二维数组。二维数组常用于表示表，表中的信息以行和列的形式组织，第一个下标代表元素所在的行，第二个下标代表元素所在的列。

1. 二维数组的创建

声明二维数组的方法有两种，语法格式分别如下：

```
数组元素类型 数组名字[][];
```

```
数组元素类型 [][] 数组名字;
```

- ☒ 数组元素类型：决定了数组的数据类型，它可以是 Java 中任意的数据类型，包括基本数据类型和非基本数据类型。
- ☒ 数组名字：为一个合法的标识符。
- ☒ 符号“[]”：指明该变量是一个数组类型变量，两个“[]”表示要创建的数组是二维数组。

【例 5.5】 创建二维数组。

```
int myarr[][];
```

与一维数组一样，如果二维数组在声明时没有分配内存空间，同样也要使用关键字 new 来分配内存，然后才可以访问每个元素。

二维数组可以看成是由多个一维数组所组成，在给二维数组分配内存时，可以为这些一维数组同时分配相同的内存。第一个中括号中的数字是一维数组的个数，第二个中括号中的数字是这些一维数组的长度。

【例 5.6】 为二维数组每一维分配相同内存。

```
a = new int[2][4];
```

为二维数组分配内存，分配后二维数组 a 拥有两个长度是 4 的一维数组。分配的内存如图 5.6 所示。

给二维数组分配内存时，还可以对其每一个一维数组单独分配内存，且分配的内存可以并不相同，在第一个中括号中定义一维数组的个数，然后就利用一维数组分配内存的方法分配内存。

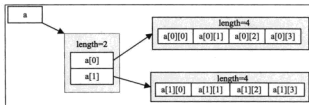


图 5.6 为二维数组分配内存

【例 5.7】 分别为每一维单独分配内存。

```
a = new int[2];
a[0] = new int[2];
a[1] = new int[3];
```

为二维数组分配内存，先给二维数组分配两个一维数组，然后对第一个一维数组分配内存 2；第二个一维数组分配内存 3。分配的内存如图 5.7 所示。

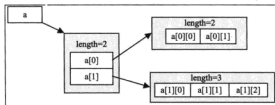


图 5.7 为二维数组分配内存

2. 二维数组初始化

二维数组的初始化与一维数组初始化类似，同样可以使用大括号完成二维数组的初始化。语法规则如下：

```
type arrayname[][] = {value1,value2...valuen};
```

- ☑ type: 数组数据类型。
- ☑ arrayname: 数组名称，一个合法的标识符。
- ☑ value: 数组中各元素的值。

【例 5.8】 初始化二维数组。

```
int myarr[][] = {{12,0},{45,10}};
```

初始化二维数组后，要明确数组的下标都是从 0 开始。如上面的代码中 `myarr[1][1]` 的值为 10。
int 型二维数组是以 `int a[][]` 来定义的，所以可以直接给 `a[x][y]` 赋值。如给 `a[1]` 的第二个元素赋值：

```
a[1][1] = 20;
```

【例 5.9】 在项目创建 Matrix 类，在主方法中编写代码实现输出一个 3 行 4 列且所有元素都是 0 的矩阵。

```

public class Matrix {                                //创建类
    public static void main(String[] args) {          //主方法
        int a[][] = new int[3][4];                  //定义二维数组
        System.out.println("输出 3 行 4 列的数组: ");
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++) {    //循环遍历数组中的每个元素
                System.out.print(a[i][j]+" ");        //将数组中的元素输出
            }
            System.out.println();                      //输出空格
        }
    }
}

```

运行结果如图 5.8 所示。

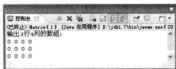


图 5.8 输出 3 行 4 列的矩阵

说明

对于整型二维数组，创建成功之后系统会赋给数组中每个元素初始值 0。

5.3.2 三维数组

三维数组的声明与一、二维数组类似，一维使用一个中括号、二维使用两个中括号，依此类推，三维则使用 3 个中括号；初始化三维数组时，由 3 层大括号进行初始化；使用时也更麻烦一些，需要使用 3 层 for 循环。

【例 5.10】 在项目中创建 Ransack 类，在类的主方法中创建三维数组，并将三维数组在控制台输出。（实例位置：光盘\TM\5\3）

```

public class Ransack {                                //创建类
    public static void main(String[] args) {          //创建三维数组
        int arr[][][] = new int[3][3][3] {
            {{ 1, 2, 3 }, { 4, 5, 6 }},
            {{ 7, 8, 9 }, {10, 11, 12}},
            {{13, 14, 15}, {16, 17, 18}}
        };
        for (int i = 0; i < arr.length; i++) {        //遍历数组
            System.out.println("三维数组的第"+(i+1)+"个元素是一个"+arr[0].length
                +"维数组，内容如下: ");
            for (int j = 0; j < arr[0].length; j++) {    //遍历数组
                for (int k = 0; k < arr[0][0].length; k++) {

```



```

        System.out.print(arr[0][k] + " ");           //输出数组元素
    }
    System.out.println();
}
}
}

```

运行结果如图 5.9 所示。

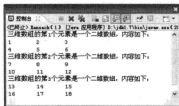


图 5.9 使用三维数组

三维数组的元素是二维数组，二维数组的元素是一维数组，每个元素使用 `arr[x][y][z]` 的格式表示，如 `arr[0][1][2]` 就代表数 6。

5.3.3 范例 3：对矩阵进行转置运算

所谓矩阵的转置就是将矩阵的行列互换，把 `a[i][j]` 的值存储在元素 `a[j][i]` 的位置，转换前是 $m \times n$ 维矩阵，转换后则是 $n \times m$ 维矩阵。运行结果如图 5.10 所示。（实例位置：光盘\TM\5\54）

在项目中创建 `ArrayRowColumnSwap` 类，在类的主方法中声明二维矩阵，然后将此矩阵进行转置运算，并将结果在控制台输出。代码如下：

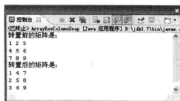


图 5.10 矩阵的转置

```

public class ArrayRowColumnSwap {                //创建类
    public static void main(String[] args) {
        int arr[][] = new int[][] { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } }; //创建二维数组
        System.out.println("转置前的矩阵是：");
        printArray(arr);                          //输出二维数组
        int arr2[][] = new int[arr.length][arr.length];
        for (int i = 0; i < arr.length; i++) {     //调整数组行列数据
            for (int j = 0; j < arr[0].length; j++) {
                arr2[j][i] = arr[i][j];
            }
        }
        System.out.println("转置后的矩阵是：");
        printArray(arr2);                          //输出二维数组
    }
}

```

```

private static void printArray(int[] arr) {
    for (int i = 0; i < arr.length; i++) {           //遍历数组
        for (int j = 0; j < arr.length; j++) {
            System.out.print(arr[i][j] + " ");       //输出数组元素
        }
        System.out.println();
    }
}

```

5.3.4 范例 4：求方阵的迹

方阵的迹，即方阵主对角线上所有元素的和，英文名是 Trace of the matrix。迹同时也是方阵的所有特征值之和，求法有很多。下面通过 Java 语言编写程序求方阵的迹。运行结果如图 5.11 所示。（实例位置：光盘\TM\5\5.5）

在项目中创建 Trace 类，在类的主方法中创建二维方阵，然后求出二维方阵的迹，并在控制台将结果输出。代码如下：

```

public class Trace {                                //创建类
    public static void main(String[] args) {
        int arr[][] = new int[][] { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } }; //创建二维数组
        int tr = 0;
        System.out.println("方阵 arr[][] 是: ");
        for (int i = 0; i < arr.length; i++) {       //遍历数组
            for (int j = 0; j < arr.length; j++) {
                System.out.print(arr[i][j] + " "); //输出数组元素
            }
            System.out.println();
        }
        for (int i = 0; i < arr.length; i++) {
            tr += arr[i][i];
        }
        System.out.println("方阵 arr[][] 的迹是: " + tr);
    }
}

```

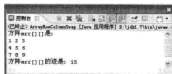


图 5.11 求方阵的迹

5.4 数组的基本操作

 视频讲解：光盘\TM\5\5\数组的基本操作.exe

java.util 包的 Arrays 类包含用来操作数组（如排序和搜索）的各种方法。本节将向读者介绍数组的

基本操作。

5.4.1 遍历数组

遍历数组就是获取数组中的每个元素。通常遍历数组都是使用 for 循环来实现。

1. 遍历一维数组

【例 5.11】 在项目中创建 GetDay 类，在主方法中创建 int 型数组，并实现将各月的天数输出。

```
public class GetDay {                                //创建类
    public static void main(String[] args) {          //主方法
        int day[] = new int[] { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }; //创建并初始化一维数组
        for (int i = 0; i < 12; i++) {                //利用循环将信息输出
            System.out.print((i + 1) + "月有" + day[i] + "天" + "\t"); //输出信息
            if ((i + 1) % 3 == 0) {                    //i+1 和 3 求余为零
                System.out.print("\n");                //输出回车
            }
        }
    }
}
```

运行结果如图 5.12 所示。

2. 遍历二维数组

遍历二维数组要比遍历一维数组麻烦一些，需使用双层 for 循环，还要通过数组的 length 属性获得数组的长度。

【例 5.12】 在项目中创建 Trap 类，在类的主方法中编写代码，定义二维数组，实现将二维数组中的元素输出。

```
public class Trap {                                //创建类
    public static void main(String[] args) {          //主方法
        int b[][] = new int[][] { { 1 }, { 2, 3 }, { 4, 5, 6 } }; //定义二维数组
        System.out.println("二维数组是: ");
        for (int k = 0; k < b.length; k++) {          //循环遍历二维数组中的每个元素
            for (int c = 0; c < b[k].length; c++) {    //将数组中的元素输出
                System.out.print(b[k][c] + " ");
            }
            System.out.println();                    //换行
        }
    }
}
```

运行结果如图 5.13 所示。

在遍历数组时，使用 foreach 语句更简单，该语句并不是一个新的语法，而是 for 循环的简化格式。下面通过 foreach 语句遍历二维数组。

【例 5.13】 在项目中创建 Tautog 类，在主方法中定义二维数组，使用 foreach 语句遍历二维数组。

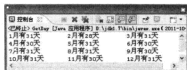


图 5.12 输出每月的天数

```

public class Tautog {
    public static void main(String[] args) {
        int arr2[][] = {{ 3, 4, 3 }, { 1, 2 }};
        System.out.println("二维数组中的元素是: ");
        for (int x[] : arr2) {
            for (int e : x) {
                System.out.print(e + " ");
            }
            System.out.println();
        }
    }
}

```

//创建类
//主方法
//定义二维数组
//提示信息
//外层循环变量为一维数组
//循环遍历每一个数组元素
//输出二维数组的元素

运行结果如图 5.14 所示。

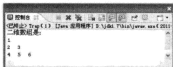


图 5.13 遍历二维数组



图 5.14 用 foreach 语句遍历二维数组

5.4.2 填充替换数组元素

数组中的元素定义完成后，可通过 Arrays 类的静态方法 fill() 来对数组中的元素进行替换。该方法通过各种重载形式可完成任意类型的数组元素的替换。fill() 方法有两种参数类型。下面以 int 型数组为例介绍 fill() 方法的使用方法。

1. fill(int[] a, int value) 方法

该方法可将指定的 int 值分配给 int 型数组的每个元素。语法格式如下：

```
fill(int[] a, int value)
```

- ☑ a: 要进行元素替换的数组。
- ☑ value: 要存储数组中所有元素的值。
- ☑ 返回值: 填充后的数组。

【例 5.14】 在项目中创建 Swap 类，在主方法中创建一维数组，并实现通过 fill() 方法填充数组元素，最后将数组中的各个元素输出。

```

import java.util.Arrays;
public class Swap {
    public static void main(String[] args) {
        int arr[] = new int[5];
        Arrays.fill(arr, 8);
        for (int i = 0; i < arr.length; i++) {

```

//导入 java.util.Arrays 类
//创建类
//主方法
//创建 int 型数组
//使用同一个值对数组进行填充
//循环遍历数组中的元素

```

        System.out.println("第"+(i+1)+"个元素是: "+arr[i]); //将数组中的元素依次输出
    }
}

```

运行结果如图 5.15 所示。

2. fill(int[] a,int fromIndex,int toIndex,int value)方法

该方法将指定的 int 值分配给 int 型数组指定范围中的每个元素。填充的范围从索引 fromIndex (包括) 一直到索引 toIndex (不包括)。如果 fromIndex == toIndex, 则填充范围为空。语法格式如下:

```
fill(int[] a,int fromIndex,int toIndex,int value)
```

- ☑ a: 要进行填充的数组。
- ☑ fromIndex: 要使用指定值填充的第一个元素的索引 (包括)。
- ☑ toIndex: 要使用指定值填充的最后一个元素的索引 (不包括)。
- ☑ value: 要存储数组所有元素的值。
- ☑ 返回值: 替换元素后的数组。



图 5.15 数组的填充

注意 如果指定的索引位置大于或等于要进行填充的数组的长度, 则会报出 `ArrayIndexOutOfBoundsException` (数组越界异常, 关于异常的知识将在后面的章节讲解) 异常。

【例 5.15】 在项目中创建 `Displace` 类, 创建一维数组, 并通过 `fill()` 方法替换数组元素, 最后将数组中的各个元素输出。(实例位置: 光盘\TM\sl\5\6)

```

import java.util.Arrays; //导入 java.util.Arrays 类
public class Displace { //创建类
    public static void main(String[] args) { //主方法
        int arr[] = new int[] { 45, 12, 2, 10, 1}; //定义并初始化 int 型数组 arr
        Arrays.fill(arr, 1, 3, 8); //使用 fill()方法对数组进行初始化
        for (int i = 0; i < arr.length; i++) { //循环遍历数组中元素
            System.out.println("第"+i+"个元素是: "+arr[i]); //将数组中的每个元素输出
        }
    }
}

```

运行结果如图 5.16 所示。

5.4.3 对数组进行排序

通过 `Arrays` 类的静态 `sort()` 方法可实现对数组排序, `sort()` 方法提供了许多种重载形式, 可对任意类型数组进行升序排序。语法格式如下:

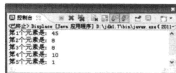


图 5.16 替换数组中元素

`Arrays.sort(object)`

☑ `object`: 指进行排序的数组名称。

☑ 返回值: 排序后的数组。

【例 5.16】 在项目中创建 `Taxis` 类, 在主方法中创建一维数组, 并将数组排序后输出。(实例位置: 光盘\TM\sl\5\7)

```
import java.util.Arrays;           //导入 java.util.Arrays 类
public class Taxis {               //创建类
    public static void main(String[] args) { //主方法
        int arr[] = new int[] { 23, 42, 12, 8, 5, 10 }; //声明数组
        System.out.println("原一维数组是: ");
        for (int i = 0; i < arr.length; i++) { //循环遍历数组
            System.out.print(arr[i] + " "); //输出原来数组
        }
        Arrays.sort(arr);           //将数组进行排序
        System.out.println("\n 升序排列后的数组是: ");
        for (int i = 0; i < arr.length; i++) { //循环遍历排序后的数组
            System.out.print(arr[i] + " "); //将排序后数组中的各个元素输出
        }
    }
}
```

运行结果如图 5.17 所示。

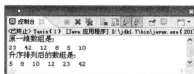


图 5.17 数组的排序



说明

Java 语言中的 `String` 类型数组的排序算法是根据字典编排序的, 因此数字排在字母前面, 大写字母排在小写字母前面。

5.4.4 复制数组

`Arrays` 类的 `copyOf()` 方法与 `copyOfRange()` 方法可实现对数组的复制。`copyOf()` 方法是复制数组至指定长度, `copyOfRange()` 方法则将指定数组的指定长度复制到一个新数组中。

1. `copyOf()` 方法

该方法提供了很多种重载形式, 来满足不同类型数组的复制。语法格式如下:

```
copyOf(arr, int newlength)
```

- ☑ arr: 要进行复制的数组。
- ☑ newlength: int 型常量, 指复制后的新数组的长度。如果新数组的长度大于数组 arr 的长度, 则用 0 填充 (根据复制数组的类型来决定填充的值, 整型数组用 0 填充, char 型数组则会使用 null 来填充); 如果复制后的数组长度小于数组 arr 的长度, 则会从数组 arr 的第一个元素开始截取至满足新数组长度为止。
- ☑ 返回值: 复制后的数组。

【例 5.17】 在项目中创建 Cope 类, 在主方法中创建一维数组, 实现将此数组复制得到一个长度为 5 的新数组, 并将新数组输出。(实例位置: 光盘\TM\sl\5\8)

```
import java.util.Arrays;
public class Cope {
    public static void main(String[] args) {
        int arr[] = new int[] { 23, 42, 12 };
        System.out.println("复制后的数组是: ");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i]+" ");
        }
        int newarr[] = Arrays.copyOf(arr, 5);
        System.out.println("\n复制后的数组是: ");
        for (int i = 0; i < newarr.length; i++) {
            System.out.print(newarr[i]+" ");
        }
    }
}
```

//创建类
//主方法
//定义数组

//循环遍历数组
//将原来数组输出

//复制数组 arr

//循环变量复制后的新数组
//将新数组输出

运行结果如图 5.18 所示。

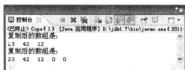


图 5.18 复制数组

2. copyOfRange()方法

该方法同样提供了多种重载形式。语法格式如下:

```
copyOfRange(arr,int formIndex,int toIndex)
```

- ☑ arr: 要进行复制的数组对象。
- ☑ formIndex: 指定开始复制数组的索引位置。formIndex 必须在 0 至整个数组的长度之间。新数组包括索引是 formIndex 的元素。
- ☑ toIndex: 要复制范围的最后索引位置。可以大于数组 arr 的长度。新数组不包括索引是 toIndex 的元素。
- ☑ 返回值: 复制指定位置后的数组。

【例 5.18】 在项目中创建 Repeat 类，在主方法中创建一维数组，并将数组中索引位置为 0~3 之间的元素复制到新数组中，最后将新数组输出。（实例位置：光盘\TM\sl\5\9）

```
import java.util.Arrays;
public class Repeat {
    public static void main(String[] args) {
        int arr[] = new int[] { 23, 42, 12, 84, 10 };
        System.out.println("原来的数组是：");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        int newarr[] = Arrays.copyOfRange(arr, 0, 3);
        System.out.println("\n 将数组下标 0~3 复制到新数组中：");
        for (int i = 0; i < newarr.length; i++) {
            System.out.print(newarr[i] + " ");
        }
    }
}
```

//创建类
//主方法
//定义数组
//输出原来的数组
//复制数组
//循环遍历复制后的新数组
//将新数组中的每个元素输出

运行结果如图 5.19 所示。

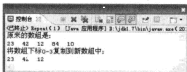


图 5.19 复制指定位置的数组

5.4.5 范例 5：对比一维、二维数组所占内存

很多程序员认为，存储相同的数据，一维数组和二维数组所占用的内存空间基本相同，这种认识是错误的，下面通过实例进行说明。运行结果如图 5.20 所示。（实例位置：光盘\TM\sl\5\10）

在项目中创建 OneArrayMemory 类，在类的主方法中使用一维和二维数组存储相同的数据，在控制台输出它们所占的内存，并加以比较。代码如下：

```
public class OneArrayMemory {
    public static void main(String[] args) {
        int num1 = 1024 * 1024 * 2;
        int[] arr1 = new int[num1];
        for (int i = 0; i < arr1.length; i++) {
            arr1[i] = i;
        }
        //获得占用内存总数，并将单位转换为 MB
        long memory1 = Runtime.getRuntime().totalMemory() / 1024 / 1024;
```

//数组中元素的个数
//创建 int 型一维数组
//为数组元素赋值

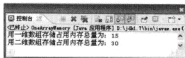


图 5.20 对比一、二维数组所占内存


```

System.out.println("用一维数组存储占用内存总量为: " + memory1);
int num2 = 1024 * 1024; //数组中元素的个数
int[][] arr2 = new int[num2][2]; //创建 int 型二维数组
for (int i = 0; i < arr2.length; i++) {
    arr2[i][0] = i; //为数组元素赋值
    arr2[i][1] = i; //为数组元素赋值
}
//获得占用内存总数, 并将单位转换为 MB
long memory2 = Runtime.getRuntime().totalMemory() / 1024 / 1024;
System.out.println("用二维数组存储占用内存总量为: " + memory2);
}
}

```

5.4.6 范例 6: 使用直接插入排序法排序

插入排序的基本思想是要将排序的值逐个插入到其所适合的位置, 其后的元素都向后移一个位置。如 1、3 和 2 排序, 先是 1, 然后 3 插入到 1 后面, 最后 2 插入到 1 后面, 3 往后移一位。运行结果如图 5.21 所示。(实例位置: 光盘\TM\sl5\11)

在项目中创建 InsertSort 类, 在类的主方法中创建一维数组, 然后将数组中的元素进行排序, 并在控制台将结果输出。代码如下:

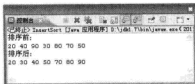


图 5.21 使用直接插入排序法排序

```

public class InsertSort {
    public static void main(String[] args) {
        int[] array = new int[] {20, 40, 90, 30, 80, 70, 50};
        System.out.println("排序前: ");
        for (int i = 0; i < array.length; i++) { //初始化数组元素
            System.out.print(array[i] + " "); //输出数组元素
        }
        int tmp; //定义临时变量
        int j;
        for (int i = 1; i < array.length; i++) {
            tmp = array[i]; //保存临时变量
            for (j = i - 1; j >= 0 && array[j] > tmp; j--) { //数组元素交换
                array[j + 1] = array[j];
            }
            array[j + 1] = tmp; //在排序位置插入数据
        }
        System.out.println("\n 排序后: ");
        for (int i = 0; i < array.length; i++) { //初始化数组元素
            System.out.print(array[i] + " "); //输出数组元素
        }
    }
}

```

5.5 经典范例

5.5.1 经典范例 1：使用冒泡排序法排序

 视频讲解：光盘\TM\lx\5\使用冒泡排序法排序.exe

冒泡排序的基本思想是对比相邻的元素值，如果满足条件就交换元素值，把较小的元素移动到数组前面，把大的元素移动到数组后面（也就是交换两个元素的位置），这样数组元素就像气泡一样从底部上升到顶部。运行结果如图 5.22 所示。

（实例位置：光盘\TM\lx\5\12）

在项目中创建 BubbleSort 类，在类的主方法中创建一维数组，使用冒泡排序法进行排序，并将结果在控制台输出。代码如下：

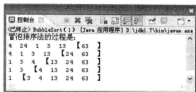


图 5.22 使用冒泡排序法排序

```
public class BubbleSort {
    public static void main(String[] args) {
        int[] array = new int[] {63, 4, 24, 1, 3, 13}; //声明并初始化一维数组
        System.out.println("冒泡排序法的过程是：");
        for (int i = 1; i < array.length; i++) {
            for (int j = 0; j < array.length - i; j++) { //比较相邻两个元素，较大的数往后冒泡
                if (array[j] > array[j + 1]) {
                    int temp = array[j]; //把第一个元素值保持到临时变量中
                    array[j] = array[j + 1]; //把第二个元素值保存到第一个元素单元中
                    array[j + 1] = temp; //把第一个元素原值保存到第二个元素中
                }
                System.out.print(array[j] + " "); //把排序后的数组元素显示到文本域中
            }
            System.out.print("【】");
            for (int j = array.length - i; j < array.length; j++) {
                System.out.print(array[j] + " "); //把排序后的数组元素显示到文本域中
            }
            System.out.println("】");
        }
    }
}
```



说明

冒泡排序法的算法比较简单，排序的结果稳定，缺点是时间效率不太高。

5.5.2 经典范例 2：输出九宫格

 视频讲解：光盘\TM\lx\5\输出九宫格.exe

九宫格是大家都喜欢玩的一种游戏，在一个三维方阵的 9 个元素中分别填入 1~9 中的 9 个数，使得每一行、列和对角线上 3 个数的和都等于 15。运行结果如图 5.23 所示。（实例位置：光盘\TM\lx\5\13）

在项目中创建 NineTable 类，在类的主方法中创建三维方阵，按照九宫格的玩法填入 1~9，在控制台将结果输出。代码如下：

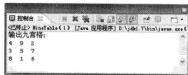


图 5.23 输出九宫格

```
public class NineTable {
    public static void main(String[] args) {
        int arr[][] = new int[3][3];
        int a = 2;
        int b = 3/2;
        for(int i=1;i<=9;i++){
            arr[a++][b++] = i;
            if(i%3==0){
                a = a-2;
                b = b-1;
            }else{
                a = a%3;
                b = b%3;
            }
        }
        System.out.println("输出九宫格: ");
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++){
                System.out.print(arr[i][j]+" ");
            }
            System.out.print("\n");
        }
    }
}
```

//创建一个三阶方阵
//第 3 行的行下标
//第 2 列的列下标
//给数组赋值
//如果 i 是 3 的倍数
//如果 i 不是 3 的倍数
//遍历输出九宫格

5.6 本章小结

本章向读者介绍的是数组的创建及使用方法。需要读者注意的是，数组的下标是从 0 开始，最后一个元素的下标总是“数组名.length-1”。本章的重点是遍历数组以及使用 Arrays 类中的各种方法对数组进行操作，如填充替换数组、复制数组等。此外 Arrays 类还提供了其他操作数组的方法，有兴趣的

读者可以查阅相关资料。

5.7 实战练习

1. 编写 Java 程序，创建一维数组 `arr[]`，并将其遍历输出。（答案位置：光盘\TM\sl\5\14）
2. 编写 Java 程序，创建一维数组 `arr[]`，将数组中最大的数输出。（答案位置：光盘\TM\sl\5\15）
3. 编写 Java 程序，创建二维数组 `arr[][]`，将二维数组中所有元素的和输出。（答案位置：光盘\TM\sl\5\16）

