第5章 数组与参数数组

从理论上说，掌握了程序执行的3种结构(顺序结构(顺序执行)，循环结构(for 、while循环)和选择结构(if、switch 语句))之后就能够编写所有的程序了，但实践起来仍然比较困难。因为程序的组成除了算法还需要数据结构，关于数据的存储是要有规划的，这会让数据的读/写更加容易。本章将介绍一种数据结构——数组。**数组**(array)是一个无序的元素序列，所有的元素类型相同，可以通过整数索引访问任何一个元素，就像数学中的数列一样。有了数组，用户就可以方便地顺序存入大量相同类型的数据，并且可以通过索引快速地取出数据。

5.1 数　　组

5.1.1 一维数组

一维数组(one-dimension arrays)是数据线性排列的数组。在使用一维数组时和普通变量一样，首先要声明一个数组变量。先写类型名称，再输入一对中括号，最后写数组变量的名称。形如：

```
int [] array;
```

在第2章中已经学习了值类型和引用类型的区别。数组是引用类型，因此刚才所做的数组声明只是在栈中开辟了一个引用地址，真正用来存储数组的数据空间还未开辟，之后就要进行数组实例的创建。首先写关键字 new，后跟元素类型名称，最后输入一对中括号，中括号内写上所创建数组的大小。形如：

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

当然，数组变量的创建和实例的创建可以写成一条语句：

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

通过实例的创建，堆中的数组空间才算是真正分配好了，栈中的引用地址也指向了堆中的数组元素。这条语句分配了一个具有10个元素的 int 类型数组，即这个数组可容纳10个 int 类型的变量。这10个变量的索引是从0到9，而不是从1到10。在计算机世界中，第1个元素往往从0开始记起，和我们平时的习惯不同。若要取数组的第5个元素，即索引值为4的元素，使用一对中括号并在中间写入4即可。形如：

```
int b = array[4];
```

到这里数组已经彻底创建好了，那么如何向其中存入数据呢？其实数组初始化的方法有很多，下面来一一介绍。

在实例创建语句的最后添加大括号，在其中写入初始化的值即可。例如：

```
int[] array = new int[5]{1,3,5,7,9};
```

这样就对数组的5个元素进行了初始化。array[0]为1，array[1]为3，array[2]为5，array[3]为7，array[4]为9。注意，若在实例的创建中指定了数组的大小，那么初始化的值必须和数组大小匹配才行。即array数组的大小是5，那么5个元素都要进行初始化。如果写成了“int [] array = new int [5] {1,3,5};”，则初始化个数与数组大小不符，不能通过编译。

在实例的创建中也可以不指定数组大小而对数组元素直接赋值。形如：

```
int[] b = new int[]{1,3,5,7};
```

数组的大小被省略，但可从赋值元素的个数求得。在该示例中数组b的大小为4，即共有4个元素。

除此之外，下面也是省略数组大小直接赋值的方法：

```
int[] c ={1,3,5,7};
```

对于数组的单个元素，可以将它们当成普通变量处理。例如要给数组的第i个变量赋值100，可以写成“a[i-1]=100;”。

下面给出一个一维数组的简单实例：求斐波那契数列。斐波那契数列是“1,1,2,3,5,8,13…”，符合F(n)=F(n-1)+F(n-2)的规律。

```
//例程5-1
static void Main(string[] args)
{
    int[] a = new int[10000];
    a[0] = 0;
    a[1] = 1;
    for (int i = 2; i <= 10000; i++)
        a[i] = a[i - 1] + a[i - 2];          //求后续斐波那契数列的值
    for (int i = 1; i <= 10000; i++)
        Console.WriteLine(a[i]);             //输出斐波那契数列
    Console.ReadLine();
}
```

由于斐波那契数列满足前两项之和等于下一项的规律，因此首先应该给出数列的前两项，先对a[0]、a[1]赋值，之后通过循环迭代求得后续值。

5.1.2 多维数组

可以将一维数组想象成逻辑上线性排列成一行的同一类型变量的集合。二维数组可以理解成若干行、若干列的同一类型变量的集合，逻辑上是一个长方形。三维数组的逻辑视图是一个长方体，以此类推。可以将矩阵的信息作为二维数组来保存，例如：

```
int [ ,] a = new int [2,3];
```

二维数组需要两个索引值，它们之间用逗号隔开。二维数组 a 有两行，每行有 3 个元素，共 56 个元素。用户在使用某一变量时也应写明两个索引值。a[i+1,j+1]表示二维数组的第 i 行中的第 j 个元素。这种二维数组每行的元素个数相同，属于长方形状，它的逻辑视图如图 5-1 所示。

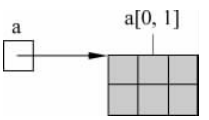

图 5-1　长方形状的二维数组的逻辑视图

二维数组实际上是由若干个一维数组组成的，每一行都可以看成是一个一维数组。二维数组还可以声明为锯齿状，即每一行的元素个数都不同。

```
int [][] a = new int[2][];                    //表示二维数组有两行
a[0] = new int [3];                           //第 1 行有 3 个元素
a[1] = new int[4];                            //第 2 行有 4 个元素
```

这样创建的二维数组的逻辑视图如图 5-2 所示。

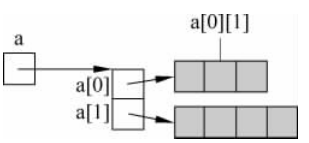

图 5-2　锯齿状的二维数组的逻辑视图

将 a[0]和 a[1]看成一维数组就好理解了。这 3 条语句先声明了数组有几行，注意和长方形状的区别，这里的二维声明使用了两对中括号，并且第 2 对中括号不写大小；而长方形状的声明是一对中括号内写两个数，之间用逗号隔开，形如[,]。请读者思考这两种形式的不同，以及为何要使用两种方法。在声明有几行后，再依次将各行视为一维数组进行实例的创建。

下面介绍二维数组的初始化。

对于长方形状的二维数组可以直接初始化，形如：

```
int[,] a = {{1,2,3},{4,5,6}};
```

和一维数组一样，这同样需要将所有变量进行初始化。和一维数组不同的是，对于每一行的数据需要用大括号，各行之间用逗号隔开。

对于锯齿状的二维数组需要逐行进行赋值，形如：

```
int [][] a = new int[2][];
a[0] = new int[]{1,2,3};
a[1] = new int[]{4,5,6,7};
```

原因显而易见，每行元素的个数不同，不能采用统一的赋值方式。

5.1.3 数组的属性

用户可以获取数组的长度。对于一维数组，通过“数组名.Length”就可以得到数组的长度。例如：

```
int[] a = new int[3];
Console.WriteLine(a.Length);
```

其输出结果是 3。

对于锯齿状的二维数组，情况就不一样了：

```
int[][] b = new int[3][];
b[0] = new int[4];
Console.WriteLine("{0}, {1}", b.Length, b[0].Length);
```

b.Length 实际上获得的是二维数组的行数；而 b[0].Length 获得的是第 0 行的一维数组元素个数，因此程序的输出是“3,4”。

由于长方形状的二维数组每行的元素个数都相同，长度信息的获取又有不同：

```
int[,] c = new int[3, 4];
Console.WriteLine(c.Length);
Console.WriteLine("{0}, {1}", c.GetLength(0), c.GetLength(1));
```

一旦数组的实例被创建，它的长度可以直接计算出来。因此 c.Length 的结果为 12，也就是所有行包括的元素个数。用户还可以获取数组的行数与列数（各维的大小），通过 c.GetLength(0)可以得到第 0 维的大小，也就是二维数组的行数；通过 c.GetLength(1)可以获取二维数组的列数。因此，第 2 条输出语句的结果为 3,4。

除此之外，System.Array 类还有很多实用的类操作，例如数组间的复制、数组的排序等。

```
int[] a = {7, 2, 5};
int[] b = new int[2];
Array.Copy(a, b, 2);
Array.Sort(b);
```

Array.Copy 的第 1 个参数是源数组，第 2 个参数是目标数组，第 3 个参数是要复制的元素个数。这个方法实现了从源数组复制指定个数的元素到目标数组。在实例中，“Array.Copy(a,b,2);”实现了将 a[0]和 a[1]两个元素复制到数组 b 中。Array.Sort 可以将数组进行从大到小的排列，请看例程 5-2。

```
//例程 5-2
static void Main(string[] args)
{
    int[] a = new int[3];
    a[0] = 5;
    a[1] = 3;
    a[2] = 1;
    Array.Sort(a);
    for (int i = 0; i < a.Length; i++)
```

```
        Console.WriteLine(a[i]);
}
```

经过排列，数组输出的结果如下：


```
1
3
5
```

5.1.4 变长数组

变长数组(variable-length arrays)在创建实例时无须指定它的大小，用户可以动态地向数组添加元素。请看例程 5-3。

```
using System;
using System.Collections;

class Program
    {
        //例程 5 - 3
        static void Main(string[] args)
        {
            ArrayList a = new ArrayList();
            a.Add("Clark");
            a.Add("Delta");
            a.Add("Alpha");
            a.Sort();
            for (int i = 0; i < a.Count; i++)
                Console.WriteLine(a[i]);

        }
    }
```

注意，使用 ArrayList 要添加一个命名空间——System. Collections。如果不在程序中添加“using System. Collections;”，编译器会报错，提醒缺少 using 指令集。

ArrayList 的用法和普通集合相同，但在声明时无须指定长度。用户可以使用 Add 方法向其中添加元素，也可以使用 Sort 方法对其进行由小到大的排序。例程 5-3 的输出结果如下：

```
Alpha
Clark
Delta
```

了解了数组的相关应用，再加上顺序结构(顺序执行)、循环结构(for、while 循环)和选择结构(if、switch 语句)就可以写出更复杂的程序了。具体的训练将会在习题中体现。

5.2 参数数组

5.2.1 重载

在学习参数数组之前应先了解一下**重载**(overloading)的概念。重载是指在相同的作用

域内声明多个同名的方法，以方便方法处理多种类型的参数。例如用户经常使用的Console.WriteLine方法，它能够输出各种类型的变量，实际上就是对各种类型都进行了重载，请看它的定义，如图5-3所示。

```
public static void WriteLine();
public static void WriteLine(bool value);
public static void WriteLine(char value);
public static void WriteLine(char[] buffer);
public static void WriteLine(decimal value);
public static void WriteLine(double value);
public static void WriteLine(float value);
public static void WriteLine(int value);
public static void WriteLine(long value);
public static void WriteLine(object value);
public static void WriteLine(string value);
public static void WriteLine(uint value);
public static void WriteLine(ulong value);
public static void WriteLine(string format, object arg0);
public static void WriteLine(string format, params object[] arg);
public static void WriteLine(char[] buffer, int index, int count);
public static void WriteLine(string format, object arg0, object arg1);
public static void WriteLine(string format, object arg0, object arg1, object arg2);
public static void WriteLine(string format, object arg0, object arg1, object arg2, object arg3);
```

图5-3　Console.WriteLine的定义

对于同样的方法名，它们的参数列表不同（无论是从参数个数还是从参数类型上来看），因此可以输出各种类型的参数，这就是重载。

对于参数类型不同，重载的方法个数并不多，但要使方法能够适应不同的参数个数，重载的方法个数就无穷无尽了，因为谁也不知道这个方法的使用者到底会一次输出多少个值，难道真的要从1个参数的方法写到100个，1000个参数的方法吗？多亏了参数数组，一个方法可以接受数量可变的参数，万能方法的存在就不需要成百上千的重载了。

5.2.2　使用数组参数

用户可以将数组作为方法的参数，这样可以将数组中的所有元素都传入。例如编写一个计算一组值之和的静态方法plus就可以用一个参数来传递整个数组。请看例程5-4。

```
class Program
{
    //例程 5 - 4
    public static int plus(int [ ] param)
    {
        int result = 0;
        foreach(int i in param)
        {
            result += i;
        }
        return result;
    }
    static void Main(string[ ] args)
    {
        int[ ] a = new int[5];
        a[0] = 1;
```

```
            a[1] = 3;
            a[2] = 4;
            a[3] = 8;
            a[4] = 11;
            int result = plus(a);
            Console.WriteLine(result);
        }
    }
```

在 Main 函数中创建了一个具有 5 个元素的 int 类型数组，并对 5 个元素进行了赋值。将数组名传入方法就计算出了这 5 个元素的和。但若要计算 6 个元素、7 个元素或者更多元素的和，需要再对更多的元素进行赋值。如果使用 params 关键字声明一个参数数组，元素赋值的代码部分就可以省略。

5.2.3 使用参数数组

params 可以用作数组参数的修饰符。这里将例程 5-4 中的 plus 方法参数进行修改：

```
public static int plus(params int [] param)
{   //函数体内部分不变
    int result = 0;
    foreach(int i in param)
    {
        result += i;
    }
    return result;
}
```

在参数前加了 params 关键字，在调用该方法时就可以传递任意数量的参数：

```
int result = plus(1,3,4,8,11);
```

编译器会帮用户把上述调用转换为以下代码：

```
int[] a = new int[6];
a[0] = 1;
a[1] = 3;
a[2] = 4;
a[3] = 8;
a[4] = 11;
int result = plus(1,3,4,8,11);
```

这样做的好处是省去了数组的声明与赋值，直接将值传入方法中即可计算。那么若要计算 6 个值的和，也可以直接调用 plus：

```
int result = plus(1,3,4,8,11,15);
```

对于 params 关键字的使用需要注意以下几点：

- 多维数组不可以使用 params 关键字。
- params 关键字不是方法签名的一部分。也就是说，如果两个方法只有 params 关键字的不同，不能算是重载。

例如：如果已经声明 public static int plus(params int [] param)，则不能再声明 public static int plus(int [] param)。

- 每个方法只能有一个 params 数组参数，并且必须是最后一个参数。
  例如：public static int plus(params int [] param,int i)的声明是不正确的。
- 重载不能有歧义。
  例如：public static int plus(params int [] param)和 public static int plus(int i, params int [] param)存在歧义。当调用 plus(1,2,3)时编译器无法知道第 1 个参数是属于数组还是 int i，也就不知道调用哪个方法了。

小　　结

本章介绍了一维数组以及多维数组的声明与使用，掌握数组的使用方法对大量数据的操作十分重要。数组的使用示例已在下文列出，供读者快速回顾。

一维数组：

```
int[] a = new int[3];
int[] b = new int[] {3, 4, 5};
int[] c = {3, 4, 5};
```

二维数组(锯齿状)：

```
int[][] a = new int[2][];
a[0] = new int[] {1, 2, 3};
a[1] = new int[] {4, 5, 6, 7};
```

二维数组(长方形状)：

```
int[,] a = new int[2, 3];
int[,] b = {{1, 2, 3}, {4, 5, 6}};
int[,,] c = new int[2, 4, 2];
```

重载(overloading)：函数签名不同，参数数量不同，参数类型不同(int /string)，参数种类不同(值类型、引用类型)。返回类型和 params 不是重载的条件。

数组参数：

```
public static int plus(int [] param)
```

参数数组：

```
public static int plus( params int [] param)
int result = plus(1,3,4,8,11);
```

习　　题

习题 5-1　通过数组编程实现：求“16,4,6,1,19,4,2,25,13,39”中第二小的数。

习题 5-2　通过数组编程实现：求“16,4,6,1,19,4,2,25,13,39”中第三小的数。

习题 5-3　通过数组编程实现：求 2017 年 1 月 1 日到 2017 年 8 月 20 日经过了多少天。

习题 5-4　通过数组编程实现：输出 n 行 n 列数字阵，它总是以对角线为起点，先横着填，再竖着填。示例如下：

```
1  2  3
4  6  7
5  8  9

1   2   3   4   5
6  10  11  12  13
7  14  17  18  19
8  15  20  22  23
9  16  21  24  25
```

习题 5-5　通过数组编程实现：输出 n 行 n 列蛇形数字阵。示例如下：

```
 1  2
 4  3

 1  2  3
 6  5  4
 7  8  9

 1   2   3   4
 8   7   6   5
 9  10  11  12
16  15  14  13
```

习题 5-6　通过数组编程实现：n 个人排队打水，每个人需要的时间为 t_i，那么第 k 个人等待的时间一共是 $t_1+t_2+\cdots+t_k$。为了提高效率，请安排一个顺序，使得每个人等待时间的总和最少。示例：5 个人打水，每个人的时间依次为 1 4 3 6 9，最少时间是 50。