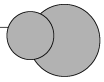
第5章

批量数据的表示与处理



在现实应用问题中，经常会用到大批量的数据，比如管理全校学生的成绩、管理图书馆的所有图书等。对于大批量数据的表示，C＃提供了数组类型。数组类型表示一组数据的集合，使用数组可以方便地定义一个名字（数组名）来表示大批量的数据（数组元素），同时数组支持通过循环结构实现批处理大量的数据。

C＃中，数组（Array）是一种包含若干变量的数据结构，这些变量都可以通过计算索引进行访问。数组中包含的变量，又称数组的元素（Element）具有相同的类型，该类型称为数组的元素类型（Element Type）。

数组类型为引用类型，数组名为实际的数组引用留出空间。实际的数组实例在运行时使用 new 运算符动态创建，该运算符自动将数组的元素初始化为它们的默认值，例如，将所有数值类型初始化为零，将所有引用类型初始化为 null。

这里先介绍几个 C＃中与数组相关的重要定义。

（1）元素：数组的独立数据项称为元素。数组的所有元素必须是相同类型的，或继承自相同类型。

（2）秩/维度：数组可以有任何为正整数的维度数。数组的维度就是秩（Rank）。

（3）维度长度：数组的每个维度都有一个长度，就是这个方向的位置个数，称为维度长度。

（4）数组长度：数组中所有维度上元素个数的总和称为数组长度。

5.1 一维数组的定义与使用

5.1.1 一维数组的定义与初始化

1. 一维数组的定义

使用数组之前必须先对其进行定义。一维数组的定义形式如下：

```
元素类型[] 数组名称=new 元素类型[数组长度];
```

或将数组定义分为两个步骤完成：

```
元素类型[] 数组名称;
数组名称=new 元素类型[数组长度];
```

在 C＃中，new 关键字可用作运算符、修饰符或约束。此处 new 关键字用作运算符，用来创建数组实例，此时会在内存中为数组申请空间。

例如：

```
int[] iSeason=new int[4];
string[] month;
month=new string[12];
```

一维数组的定义说明：

(1) 一维数组是由元素类型、数组名和长度组成的构造类型。元素类型指示存放在数组中的元素是什么类型，C＃中数组元素的类型可以是值类型或引用类型。

(2) 数组名必须符合 C＃中标识符的命名规则。

(3) 数组长度必须大于或等于 1，表示数组中元素的个数。数组在创建之前，数组长度必须已知，并且数组一旦被创建，其大小就固定了。命名空间 System.Collections 中的 ArrayList 类可实现动态数组。

(4) 数组的索引号从 0 开始，即如果维度长度为 n，索引号的范围就是 0～n－1。

2. 一维数组的存储形式

一维数组是指定类型、指定数目的元素的数据集合，其每个元素的数据类型都相同，因而元素的内存形式也是相同的。C＃中一维数组元素是连续存放的，即在内存中一个元素紧跟着另一个元素线性排列，所以一维数组的存储形式就是多个元素内存形式连续排列的结果，即一维数组的数组元素按递增索引顺序存储，从索引 0 开始，到索引数组长度－1 结束。

C＃通过数组名加相对偏移量来查找元素，相对偏移量由索引计算生成，索引表示元素在数组中的位置。最前面元素的索引值为 0，索引的值规律递增，n 个元素的数组最后一个元素的索引是 n－1。每个元素的内存长度由元素的类型确定。

3. 一维数组的初始化

当数组被创建之后，每个元素被自动初始化为元素类型的默认值。对于预定义的类型，整型默认值为 0，浮点型默认值为 0.0，布尔型默认值为 false，而引用类型的默认值为 null。

同时，C＃还支持对数组的显式初始化，即在创建数组时包括初始化列表来设置显式表达式，数组显式初始化的一般形式如下：

```
元素类型[] 数组名称=new 元素类型[数组长度]{与数组大小相等个数的元素值列表};
```

当初始化时，初值的个数一定要与数组大小相等，否则会出现编译错误。下面是正确的一维数组初始化示例：

```
int[] iSeason=new int[4]{1,2,3,4};
string[] month=new string[12]
{"Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"};
```

对一维数组进行显式初始化时,需要注意:

(1) 初始值必须有逗号分隔,并封闭在一对大括号内。

(2) 维度长度是可选的,因为编译器可以通过初始值的个数来推算出长度,例如:

```
int[] iSeason=new int[]{1,2,3,4};
string[] month=new string[]{"Jan","Feb","Mar","Apr","May","Jun","Jul","Aug",
"Sep","Oct","Nov","Dec"};
```

(3) 需要注意在数组创建表达式和初始化列表之间没有任何分隔符。

(4) 为了书写方便,一维数组的初始化还可以写成下面的简洁形式:

```
int[] iSeason={1,2,3,4};
string[] month={"Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct",
"Nov","Dec"};
```

但是,此种简洁形式只能用于数组声明的同时进行初始化。

5.1.2 一维数组元素的引用

数组必须先定义后使用,且只能逐个引用数组元素的值,不能一次性引用整个数组。数组元素的引用是通过下标得到的,一般形式为

```
数组名[下标表达式]
```

其中,一对方括号[]为下标引用运算符,见表5.1。

表5.1 下标引用运算符

类 别	运算符	用 法	功 能	结合性	示 例
一元运算符	[]	op1[op2]	下标引用	自左向右	iSeason[0]

下标引用运算符在所有运算符中优先级较高,其作用是引用数组中的指定元素,运算结果为左值(即元素本身),因此可以对运算结果做赋值、自增、自减等运算。例如:

```
int[] A=new int[4]{1,2,3,4};
int x;
x=A[0];                          //x 被赋值为 A[0],即值为 1
A[2]=10;                         //A[2]被重新赋值,数组 A 变为{1,2,10,4}
A[3]++;                          //A[3]自增,数组 A 变为{1,2,10,5}
```

使用下标引用运算符时需注意:

(1) op1 必须为数组名,op2 为下标表达式,表示数组元素的索引。下标表达式可以为常量、常量表达式、变量及变量表达式,但必须为整数,并且不能是负数。数组元素的下标从0开始,与其内存形式对应。C# 约定数组最前面的元素为第0个元素,然后是第1个、第2个……

(2) 元素的下标值不能超过数组的长度,否则会导致数组下标越界的严重错误。但是,编译时并不会出错,程序运行时会提示“索引超出了数组界限”的错误异常。

(3) 数组不允许以整体形式进行算术运算等,但是 C# 支持数组的整体赋值。

【例 5.1】 从键盘输入并逆序输出 12 个月的英文缩写。

程序代码如下:

```
using System;
class ArrayMonth
{
    static void Main()
    {
        string[] month=new string[12];  //定义数组保存 12 个月的英文缩写
        for(int i=0; i<12; i++)     //输入 12 个月的英文缩写赋给数组的相应元素
            month[i]=Convert.ToString(Console.ReadLine());
        for(int i=11; i>=0 ; i--)         //逆序输出数组中保存的 12 个月英文缩写
          Console.WriteLine(month[i]);
    }
}
```

【例 5.2】 把数组 A 的元素复制到数组 B 中。

分析:C# 支持的复制数组方法有两种,分别为逐元素复制和整体复制。

逐元素复制的程序代码如下:

```
using System;
class ArrayCopy
{
    static void Main()
    {
        int[] A=new int[] { 1, 2, 3, 4 };
        int[] B=new int[4];
        for(int i=0; i<4; i++)                  //逐元素复制
            B[i]=A[i];
        for(int i=0; i<4; i++)
            Console.WriteLine(B[i]);
    }
}
```

数组整体复制程序代码如下:

```
using System;
class ArrayCopy
{
    static void Main()
    {
        int[] A=new int[] { 1, 2, 3, 4 };
        int[] B=new int[4];
        B=A;                                    //整体复制
        for(int i=0; i<4; i++)
```

```
10              Console.WriteLine(B[i]);
11      }
12  }
```

5.2 多维数组的定义与引用

虽然进行批量数据表示时,一维数组的使用频率很高,但是在一些情况下也可能需要多维数组。C#中允许定义多维数组,多维数组的每个元素又是一个数组,称为子数组。

C#中多维数组有两种类型:矩形数组和交错数组。矩形数组是某个维度的所有子数组有相同长度的多维数组,并且不管该矩形数组有多少维度,总是使用一组方括号进行所有维度描述。交错数组是每个子数组都是独立数组的多维度数组,子数组长度可以具有不同的长度,并且数组的每个维度使用一组方括号来描述。相比交错数组,矩形数组具有较高的使用频率。

5.2.1 矩形数组的定义与初始化

1. 矩形数组的定义

矩形数组的一般定义形式如下:

```
元素类型[,,…,] 数组名称=new 元素类型[第一维数组长度, 第二维数组长度,…, 第 N 维数组长度];
```

或将数组定义分为两个步骤完成:

```
元素类型[,,…,] 数组名称;
数组名称=new 元素类型[第一维数组长度, 第二维数组长度,…, 第 N 维数组长度];
```

例如:

```
int[,] A=new int[3,3];
int[,,] B;
B=new int[2,3,4];
```

多维数组的定义说明:

(1) 多维数组声明中方括号内的逗号是秩说明符,它们指定了数组的维度数。秩就是逗号数量加1。比如,一个逗号代表二维数组,两个逗号代表三维数组,以此类推。

(2) 多维数组声明时,原则上可以使用任意多的秩说明符,即C#支持任意多维的数组。

(3) 多维数组的秩作为数组类型的一部分,而维度长度不是类型的一部分。

(4) 多维数组一旦声明,其维度数就被固定。但是,其维度长度需要在创建数组时才会被确定。

2. 矩形数组的存储形式

矩形数组的存储形式类似一维数组,也是顺序存储的方法,数组元素的存储方式是:

首先增加最右边维度的索引，然后是左边紧邻的维度，以此类推，直到最左边。原则上，可将任意维度的数组都视为线性的。

3. 矩形数组的初始化

矩形数组被创建之后，可以借助 new 运算符将每一个元素自动初始化为类型的默认值。

C＃也支持对矩形数组进行显式初始化，一般形式如下：

元素类型[,,…,] 数组名称=new 元素类型[第一维数组长度, 第二维数组长度,…, 第 *N* 维数组长度]{用大括号和逗号分隔的与数组大小相等个数的元素值列表};

例如：

```
int[,] A=new int[3,3]{{1,2,3},{4,5,6},{7,8,9}};
int[,,] B=new int[2,3,4]{{{1,2,3,4},{5,6,7,8},{9,10,11,12}},{{13,14,15,16},
{17,18,19,20},{21,22,23,24}}};
```

对多维数组进行显式初始化时，需要注意：

（1）初始值必须按照多维数组的维度和每一维的长度使用大括号进行封闭，并且每个初始值向量必须有逗号分隔，且每个初始值向量里的初始值之间也要使用逗号分隔。

（2）维度长度是可选的，因为编译器可以通过初始值的个数来推算出长度，但是表示数组秩的逗号不能省略。例如：

```
int[,] A=new int[,]{{1,2,3},{4,5,6},{7,8,9}};
```

（3）为了书写方便，多维数组的初始化还可以写成下面的简洁形式：

```
int[,] A={{1,2,3},{4,5,6},{7,8,9}};
```

但是，此种简洁形式只能用于数组声明的同时进行初始化。

5.2.2　矩形数组元素的引用

矩形数组元素的引用与一维数组相似，只能逐个引用数组元素的值，不能一次性引用整个数组。数组元素引用是通过下标得到的，一般形式为

数组名[下标表达式 1,下标表达式 2,…,下标表达式 *N*]

或

数组名[下标表达式 1][下标表达式 2]…[下标表达式 *N*]

矩形数组中元素每一维的下标相互独立，各自标识在本维度的位置。例如：

```
int[,] A=new int[3,3]{{1,2,3},{4,5,6},{7,8,9}};
int x;
x=A[0,1];                    //x 值为 2
A[2,2]=10;                   //A[2,2]被重新赋值,数组 A 变为{{1,2,3},{4,5,6},{7,8,10}}
```

【例 5.3】 从键盘给一个二维矩形数组输入信息并按行列输出。

程序代码如下：

```
using System;
class ArrayScan
{
    static void Main()
    {
        int[,] A=new int[3, 4];                    //定义二维矩形数组
        for(int i=0; i<3; i++)                     //双重循环实现二维数组元素的输入
            for(int j=0; j<4; j++)
                A[i, j]=Convert.ToInt32(Console.ReadLine());
        for(int i=0; i<3; i++)                     //双重循环实现二维数组元素的输出
        {
            for(int j=0; j<4; j++)
                Console.Write("{0,5:d}", A[i, j]);
            Console.WriteLine();                   //每输出一行之后换行
        }
    }
}
```

程序运行情况如下：

```
1↙
2↙
3↙
4↙
5↙
6↙
7↙
8↙
9↙
10↙
11↙
12↙
    1    2    3    4
    5    6    7    8
    9   10   11   12
```

5.2.3 交错数组的定义与使用

1. 交错数组的定义和初始化

交错数组是数组的数组。与矩形数组不同，交错数组的子数组可以有不同数目的元素。一般情况下，交错数组的声明和创建是分层进行的，如二维交错数组的声明方式

如下：

```
元素类型[][] 数组名称=new 元素类型[顶层数组大小][];
数组名称[0]=new 元素类型[第一个子数组的大小];
数组名称[1]=new 元素类型[第二个子数组的大小];
数组名称[2]=new 元素类型[第三个子数组的大小];
```

对交错数组进行定义的同时也可以进行显式初始化，语法与矩形数组相似。例如：

```
int[][] Arr=new int[3][];
Arr[0]=new int[]{10,20,30};
Arr[1]=new int[]{40,50,60,70};
Arr[2]=new int[]{80,90};
```

2. 交错数组元素的引用

交错数组元素的引用也是通过下标得到的，一般形式为

```
数组名[下标表达式 1][下标表达式 2]…[下标表达式 N]
```

交错数组中元素每一维的下标相互独立，各自标识数组元素在本维度的位置。例如：

```
int[][] Arr=new int[3][];
Arr[0]=new int[]{10,20,30};
Arr[1]=new int[]{40,50,60,70};
Arr[2]=new int[]{80,90};
int x;
x=Arr[0][1];   //x 值为 20
Arr[2][1]=10;  //Arr[2,2]被重新赋值,数组 Arr 变为{{10,20,30},{40,50,60,70},{80,10}}
```

【例 5.4】 交错数组的遍历。

程序代码如下：

```
1    using System;
2    class ArrayScan
3    {
4        static void Main()
5        {
6            int[][] Arr=new int[3][];
7            Arr[0]=new int[] { 10, 20, 30 };
8            Arr[1]=new int[] { 40, 50, 60, 70 };
9            Arr[2]=new int[] { 80, 90 };
10           for(int i=0; i<Arr.GetLength(0); i++)
11                    //借助 GetLength 方法获得 Arr 的维度长度,即 Arr 包括几个子数组
12           {
13               for(int j=0; j<Arr[i].GetLength(0); j++)
14                    //借助 GetLength 方法获得 Arr[i]的维度长度,即每个子数组的大小
```

```
15                  Console.Write("{0,5:d}", Arr[i][j]);
16              Console.WriteLine();
17          }
18      }
19  }
```

程序运行情况如下：

```
10   20   30
40   50   60   70
80   90
```

5.3 数组应用程序举例

5.3.1 foreach 语句

对数组进行遍历时，除了可以采用第3章学过的循环语句(如5.1节和5.2节使用的for语句)之外，这里再介绍一个对数组进行遍历时更常用的语句——foreach语句。foreach语句用于枚举一个集合的元素，此处只讨论foreach语句在数组操作中的应用。foreach语句的语法如下：

```
foreach(数据类型 变量 in 数组名)
    语句
```

使用foreach语句需要注意：

(1) foreach语句中数据类型和变量组成一个临时迭代变量声明。foreach语句使用迭代变量来连续表示数组中的每一个元素。

(2) 对临时迭代变量的声明，其数据类型可以显式地声明为数组中元素的类型，也可以使用var关键字来隐式提供它的类型，然后由编译器根据数组的类型推断临时迭代变量的类型。

(3) 迭代变量是只读的，不能被修改。这包括两层含义：第一，对于值类型的数组不能改变数组的数据；第二，对于引用类型的数组不能改变实际数据的引用，但是实际数据有可能通过迭代变量被修改。

(4) foreach语法格式中的语句是要为数组中的每个元素执行一次的简单语句或语句块。

(5) 使用foreach语句遍历数组时，不需要设置循环条件，数组遍历更简单快捷，也更安全，尤其对于多维矩形数组的遍历，只需一个foreach语句即可实现。

【例5.5】 使用foreach语句实现一维数组元素的遍历。

程序代码如下：

```
1   using System;
2   class ArrayScan
3   {
```

```
4       static void Main()
5       {
6           int[] A={ 1, 2, 3, 4 };
7           foreach(int i in A)
8               Console.WriteLine(i);
9       }
10  }
```

程序的运行情况如下：

```
1
2
3
4
```

【例 5.6】 使用 foreach 语句实现求交错数组中所有元素的和。

程序代码如下：

```
1   using System;
2   class ArraySum
3   {
4       static void Main()
5       {
6           int sum=0;
7           int[][] A=new int[2][];
8           A[0]=new int[4] { 1, 2, 3, 4 };
9           A[1]=new int[6] { 5, 6, 7, 8, 9, 10 };
10          foreach(int[] arr in A)              //处理顶层数组
11              foreach(int i in arr)            //处理二层数组
12                  sum=sum+i;
13          Console.WriteLine("ArraySum is {0}", sum);
14      }
15  }
```

程序的运行情况如下：

```
ArraySum is 55
```

5.3.2　数组应用举例

1. 查找

查找问题是批量数据处理中的一个基本问题，其目标就是在一批数据中查找指定数据，如最值查找或指定值查找，查找结束后应该给出查找成功与否的结论。

目前常用的查找算法有顺序查找、二分查找、Hash 查找、二叉排序树查找等。这里只介绍顺序查找和二分查找，其他算法请读者自行查阅相关资料。