第 5 章 数组与字符串

CHAPTER 5

本章要点：

- 一维数组的介绍；
- 二维数组的介绍；
- 交错数组的使用；
- Array 类及其方法的应用；
- 字符串及其应用；
- StringBuilder 类及其应用。

5.1 数组概述

前面已经学习了变量的使用，知道变量的作用是存储数据。现在的问题是，如果我们有多个相同类型的数据需要保存，那应该如何处理呢？难道需要使用多次变量定义？正确的答案是使用数组类型，这是 C#提供的专门用于处理大量相同类型数据的简便方法。可以这样理解：定义普通变量是申请一块内存保存一个数据，而定义数组变量则是申请一段连续的内存保存多个数据。比如定义 100 个学生，就不需要再定义 100 次学生变量了。只需要将这 100 个学生看成一个整体，定义一个数组变量即可。但在内存中，这个数组变量则包含了 100 个学生变量所占的内存区域。

数组是具有相同数据类型的一组数据的集合，例如，球类的集合有足球、篮球、羽毛球等；电器集合有电视机、洗衣机、电风扇等。前面学过的变量用来保存单个数据，而数组则保存的是多个相同类型的数据。数组与单个变量的内存区域比较如图 5.1 所示。

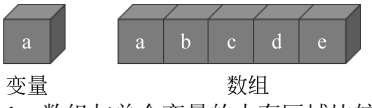

图 5.1 数组与单个变量的内存区域比较图

数组是通过指定数组的元素类型、数组的秩（维数）和数组每个维度的上限和下限来定义的。因此一个数组的定义需要包含以下几个元素：元素类型、数组的维数、每个维度的上下限。

数组的定义语法如下：

```
数据类型  数组名 = new  数据类型[数组长度]
```

例如：

```
int[] arr1 = new int[10];  //定义一个名为 arr1 的 int 型数组，其最多容纳 10 个元素。
```

对数组的定义的组成要素注解如图 5.2 所示。

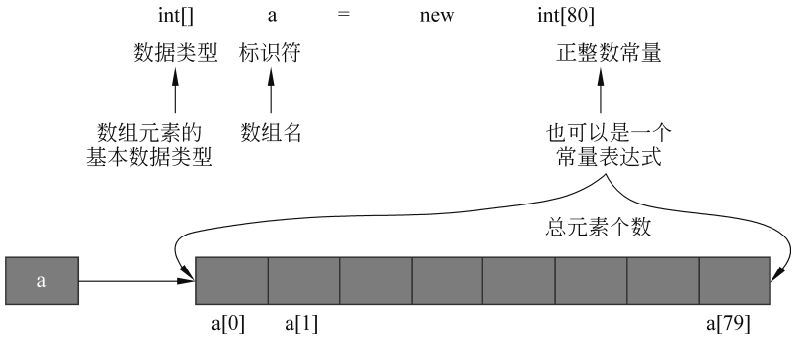

图 5.2 数组定义的组成要素

数组中的每一个变量称为数组的元素。数组能够容纳元素的数量称为数组的长度。数组中的每个元素都具有唯一的索引与其对应。数组的索引从零开始。在程序设计中引入数组，将若干个元素看作一个整体，可以更为有效地管理数据。将若干个元素看作一个数组的情况，称为一维数组。而将数组中的元素也可以为另一个数组。因此这种将数组作为元素的情况，称为二维数组，甚至多维数组等情况。下面就将对这些数组进行一一介绍。

微课视频 5-1

5.2 一维数组

一维数组实际上是一组相同数据类型的信息集合。例如，学校的学生排列的一字长队就是一个数组。每一位学生都是数组中的一个元素。本节将介绍一维数组的创建及使用。

数组作为引用类型，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。一维数组的创建有两种形式。

1. 先声明，再用 new 关键字进行内存分配

```
数据类型[]  数组名;
```

例如：

```
int[] arr1;
```

数据类型决定了数组中元素的数据类型。它可以是在 C#中任意的数据类型，数组名为一个合法的标识符符号。“[]”表示当前类型是数组，单个“[]”表示要创建的数组是一维数组。

声明数组后，还不能访问它的任何元素。因为声明数组，只是给出了数组名字和元素的数据类型。当只是声明数组之后，编译器会在内存的栈区找一个内存区域，将当前的数组名保存起来，其内存区域如图 5.3 所示。然后这块区域会保存真正存储了所有元素所在的位于内存堆上的地址。因此想要真正使用数组，还要为它分配内存空间。在为数组分配内存空间时，必须指明数组的长度，为数组分配内存空间的语法格式如下：

```
数组名 = new 数据类型[长度];
```

例如：

```
arr1 = new int[10];
```

通过上面的语法可知，使用 new 关键字分配数组时，必须指定数组元素类型和元素的

图 5.3　一维数组的内存区域

个数，即数组的长度。使用 new 关键字为数组分配内存时，整型数组中各个元素的初始值都为 0。

回忆一下：每一个 int 型的变量所占的内存大小是多少？

在上面的示例中，arr1 这一个数组分配了 10 个大小为 int 型的内存空间，且这些空间是连续的。

2. 声明的同时为数组分配内存

这种创建数组的方法是将数组的声明和内存的分配合并在一起执行。语法如下：

```
数据元素类型[]  数组名 = new  数据类型[数组长度]
```

例如：

```
int[] arr1 = new int[10];  //定义一个名为 arr1 的 int 型数组，其最多容纳 10 个元素。
```

5.2.1　一维数组初始化

数组的初始化主要分为两种：为单个元素赋值和同时为整个数组赋值。下面分别介绍。

1. 为单个元素赋值

在声明一个数组并为其指定长度后，就可以通过数组元素的下标来访问每个元素，并为这些元素赋值。例如：

```
int[] arr = new int[5]; //数组长度为 5
arr[0] = 1;  //数组下标从 0 开始，即下标为 0 的为第 1 个元素
arr[1] = 2;
arr[2] = 3;
arr[3] = 4;
arr[4] = 5; //数组元素的最大下标为 4
```

利用这种方法，可以依次为每一个元素进行赋值。但如果数组的长度较大时，则会使程序显得很冗长。如果存入数组的所有元素的值是有一定规律的，则可以使用循环来完成这一赋值过程。例如：

```
int[] arr = new int[5];
for( int i = 0; i<arr.Length; i++)
{
    arr[i] = i+1;
}
```

第 2 行代码中的 arr.Length 的作用是获得数组 arr 的长度。需要注意的是，在访问数组时，中括号中元素的下标一定是在 0~（数组长度−1）的范围内，否则会产生编译错误。

2. 同时为整个数组赋值

在声明并为数组分配内存时，可以同时完成对整个数组的赋值操作。示例如下所示：

```
int[] arr = new int[5]{1,2,3,4,5}; //声明时赋值
int[] arr = new int[]{1,2,3,4,5};  //右侧省略数组长度，以赋值个数为数组长度；
```

```
int[] arr = {1,2,3,4,5};                    //更为简略的写法
```

以上 3 种形式都可以完成整个数组的赋值操作，它们的效果是一样的，都定义了一个长度为 5 的整型数组，并进行了初始化。其中后两种编译器会自动计算赋值个数，并将之作为数组的长度。

5.2.2 一维数组的使用

一维数组的使用与一维数组的赋值比较类似，都是通过访问数组元素的下标来获取或设置元素的值。

扩展资源：实例 5.1-输出一年中每个月的天数实例。完整的代码及代码解析请扫描 5.2 节节首的二维码观看微课视频进行学习。


微课视频 5-2

5.2.3 案例 5：生成一副扑克牌

一副扑克牌一共有 52 张牌组成，其中一共有 4 种花色，每种花色有 13 个等级值。生成一副扑克牌并将其打乱，最后显示生成的扑克牌的前 4 张的牌面。

分析上述案例，可知，首先需要使用两个一维数组分别将花色和等级值保存起来；其次需要找到生成一副牌面不重复的 52 张牌的方法；之后将 52 张牌打乱，再显示前 4 张牌即可。

示例代码如代码 5.1 所示。

代码 5.1 案例 5：生成一副扑克牌示例代码

```
01   class Program
02   {
03      static void Main(string[] args)
04      {
05         int[] deck = new int[52];
06         string[] suits = { "黑桃", "红心", "梅花", "方块" };
07         string[] ranks = { "A", "2", "3", "4", "5", "6", "7", "8", "9",
     "10", "J", "Q", "K" };
08         for (int i = 0; i<deck.Length; i++)
09            deck[i] = i;
10         Random rand = new Random();
11         for(int i=0;i<deck.Length;i++)
12         {
13            int index = rand.Next(deck.Length);
14            int temp = deck[i];
15            deck[i] = deck[index];
16            deck[index] = temp;
17         }
18         for(int i=0;i<4;i++)
19         {
20            string suit = suits[deck[i] / 13];
21            string rank = ranks[deck[i] % 13];
22            Console.WriteLine($"第{i+1}张牌为{suit}{rank}");
23         }
24         Console.ReadKey();
```

```
25        }
26    }
```

上述代码首先将 deck 数组使用 0～51 的值进行填充，再使用 for 循环对 deck 数组中的值进行打乱。最后利用 deck 数组中的值，计算出对应的花色和等级值并显示出牌面。运行效果如图 5.4 所示。

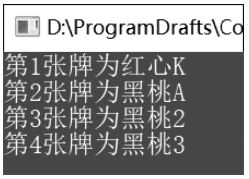

图 5.4　案例 5：生成一副扑克牌运行效果图

微课视频 5-3

5.3　二维数组

二维数组是一种特殊的多维数组。多维数组是指可以用多个索引访问的数组，声明时使用多个中括号或者在中括号内加逗号，就表明是多维数组。有 n 个中括号或者中括号内有 n 个逗号，就表示是 n+1 维数组。在 C#中，多维数组有两种情况：一种是每一行的列数是相同的，即为矩形方阵；另一种是每一行的列数可能是不相同的，即为不规则数组。这里讨论的二维数组则为矩形方阵的情况。不规则数组将会在 5.4 节中讨论。

5.3.1　二维数组的创建

二维数组常用于表示二维表，表中的信息以行和列的形式展示。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。比如一幢大楼，有若干层，每一层有若干个房间。于是就可以通过绘制一个二维表来记录每一个房间的位置。

二维数组的声明语法如下：

```
数据类型[,] 数组名;
```

例如：

```
int[,] arr2d; //声明一个名为 arr2d 的二维数组
```

与一维数组一样，二维数组在声明时也没有分配内存空间。同样也需要使用 new 关键字来完成内存的分配，然后才可以访问每个元素。对于矩形方阵的二维数组而言，常用的内存分配方式为直接分配。例如：

```
int[,] arr2d = new int[2,4]; //定义一个 2 行 4 列的二维数组
```

5.3.2　二维数组的初始化

二维数组的下标有两个，分别代表行索引和列索引，构成由行和列组成的一个矩阵，如图 5.5 所示。

二维数组的初始化有两种方式：为单个二维数组元素赋值、同时为整个二维数组赋值。

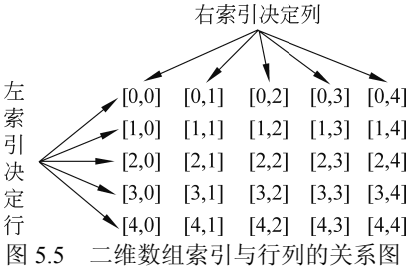

图 5.5　二维数组索引与行列的关系图

1. 为单个二维数组元素赋值

为单个二维数组元素赋值，即首先声明一个二维数组，并指定行数和列数，然后为二维数组中的每个元素进行赋值，例如：

```
int[,] myarr = new int[2,2]; //定义一个 int 类型的二维数组
myarr[0,0] = 0; //为二维数组第 1 行第 1 列赋值
myarr[0,1] = 1; //为二维数组第 1 行第 2 列赋值
//为二维数组第 2 行第 1 列赋值
myarr[1,1] = 3; //为二维数组第 2 行第 2 列赋值
```

同一维数组的赋值一样，为单个二维数组元素赋值则需要使用嵌套循环来处理。例如：

```
int[,] myarr = new int[2,3];//定义一个 int 类型的二维数组
for(int i = 0; i< 2; i++)//遍历二维数组的行
{
    for(int j = 0; j < 3; j++)//遍历二维数组的列
    {
        myarr[i,j] = i + j;//访问对应行和列的数组元素
    }
}
```

2. 同时为整个二维数组赋值

同时为整个二维数组赋值时需要使用嵌套的大括号，将要赋值的数据包含在里层大括号中，每个大括号中间用逗号隔开。例如：

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

以上 3 种形式实现的效果是一样的，都是定义了一个 2 行 2 列的 int 型二维数组，并进行了初始化。后面两种均会自动计算数组的行数和列数。

5.3.3 二维数组的使用

实际应用中，经常会获取二维数组的行数和列数。获取二维数组的行数可通过 GetLength(0)属性来获得，列数可通过调用 GetLength(1)来计算获得。

扩展资源：*实例 5.2-模拟客房预订系统实例。完整的代码及代码解析请扫描 5.3 节节首的二维码观看微课视频进行学习。*


微课视频 5-4

5.3.4 案例 6：数独游戏判定

数独是一种数字游戏，由 9×9 个格子组成。而这 9×9 个格子又被分成 9 个 3×3 的小格子。在每个 3×3 的小格子中填入 1~9 的数字，最终整个 9×9 的格子都由 1~9 的数字填满。数独一开始会随机选一些格子填入数字，要求玩家填满其他空白格。要求的是每一行、每一列、每一个 3×3 的小格子中只能有 1~9 的不重复的数字。那么如何判断玩家填入的数字是否正确呢？本案例就来讨论这一问题。

从数独的游戏来看，我们可以分析出，可以使用二维数组的方式来存储数独的 9×9 个格子中的数。默认情况下，整个数独中的数字均为 0，表示空格。当系统给出初始题面时，对应位置则赋予初值，而其他依然为 0。玩家要做的就是给出空白格子中应填入的数字。

因此可以看出，当玩家给出了答案后，对数独的判定就转变成了对二维数组的判定。判定的条件即为数独解题成功的上述 3 个条件。因此可得出参考代码如代码 5.2 所示。

代码 5.2　案例 6：数独游戏判定示例代码

```
01   class Program
02   {
03       static void Main(string[] args)
04       {
05           bool result = true;
06           int[,] grid = new int[9, 9];
07           for (int i = 0; i< 9; i++)
08               for (int j = 0; j < 9; j++)
09                   grid[i, j] = int.Parse(Console.ReadLine());
10           for (int i = 0; i< 9; i++)
11               for (int j = 0; j < 9; j++)
12               {
13                   bool isColumn=true, isRow=true, isGrid=true;
14                   for (int index = 0; index < 9; index++)
15                   {
16                      if (index != j && grid[i, index] == grid[i, j])
17                          isColumn = false;
18                      if (index != i && grid[index, j] == grid[i, j])
19                          isRow = false;
20                  }
21                  for (int row = (i / 3) * 3; row < (i / 3) * 3 + 3; row++)
22                     for (int col = (j / 3) * 3; col < (j / 3) * 3 + 3; col++)
23                        if (row != i && col != j && grid[row, col] == grid[i, j])
24                            isGrid = false;
25                  result = result && !(grid[i, j] < 1 || grid[i, j] > 9
     || !(isColumn && isRow && isGrid));
26               }
27           if (result)
28               Console.WriteLine("正确");
29           else
30               Console.WriteLine("错误");
31           Console.ReadKey();
32       }
33   }
```

上述代码利用多重循环完成对二维数组中元素的检查，最终完成判定。读者可以进一步思考，考虑如何提高判定效率。

5.4　不规则数组（交错数组）

前面讲的二维数组都是规则的矩阵形数组，即每行具有相同的列数。但实际情况是，有时候每一行的列数是不同的。这样的二维数组实际是一个不规则型的数组。不规则型数组也有二维和多维之分。定义二维不规则数组时，使用两个中括号“[][]”；如果是定义三

维不规则数组，则使用 3 个中括号“[][][]”，以此类推。以二维不规则数组为例，其定义方式如下：

```
int[][] myarr = new int[3][];//定义一个 int 类型的二维不规则数组
myarr[0] = new int[5];//为第 1 行分配 5 个元素
myarr[1] = new int[3];//为第 2 行分配 3 个元素
myarr[2] = new int[4];//为第 3 行分配 4 个元素
```

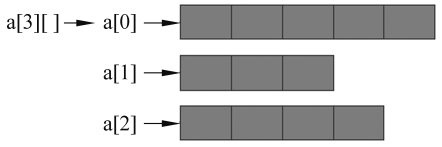

图 5.6　不规则二维数组的空间占用

因此上面代码中定义的不规则二维数组的空间占用如图 5.6 所示。

对于不规则数组而言，当每一维度的长度都相同时，也就变成了规则数组。因此规则数组上的所有操作对于不规则数组都是成立的。但对于不规则数组，在初始化时，必须为每一维数组元素进行赋值。

以不规则的二维数组为例，首先需要定义这个不规则二维数组，然后将每一行初始化为一个一维数组。例如：

```
int[][] myarr = new int[3][];//定义一个 int 类型的二维不规则数组
myarr[0] = new int[]｛0，1｝;//初始化第 1 行
myarr[1] = new int[]｛0，1，2，3｝;//初始化第 2 行
```

在不规则的数组中，由于每一维都是一个单独的数组，因此可以通过行下标来获取指定行的列维数，如：myarr[0].Length 可以获得 myarr 这个不规则数组的第 1 行的列数。例如，如代码 5.3 所示，可以通过循环为不规则数组进行初始化。

代码 5.3　通过循环为不规则数组进行初始化示例代码

```
01  using System;
02  namespace Hello_World
03  {
04      class Program
05      {
06          static void Main(string[] args)
07          {
08              int[][] arr = new int[3][]; //声明不规则数组
09              arr[0] = new int[5];
10              arr[1] = new int[3];
11              arr[2] = new int[4];
12              for(int i = 0; i<arr.Length; i++)//获取不规则数组的行数
13              {
14                 for(int j = 0; j <arr[i].Length; j++)//根据行标，获取对应行的列数
15                 {
16                      Console.Write(arr[i][j]);
17                  }
18                 Console.WriteLine();
19              }
20              Console.ReadLine();
21          }
22      }
23  }
```


微课视频 5-5

5.5　C#中的 Array 类及数组的遍历操作

5.5.1　Array 类

C#中的数组本质是一种引用类型的对象，其是由 System.Array 类派生一种类之间的关系，将在第 8 章中详细介绍而来的。在 Array 类中定义了各种属性或方法。由于数组与 Array 类是派生关系，因此根据派生的特性，数组对象是可以完全使用 Array 类中所定义的各种操作的。例如，可以使用 Array 类的 Length 属性获取数组元素的长度，也可以使用 Rank 属性获取数组的维数。Array 类的常用方法如表 5.1 所示。

表 5.1　Array 类的常用方法

方　法	说　明
Copy()	将数组中的指定元素复制到另一个目标数组中
CopyTo()	从指定的目标数组索引处开始，将当前一维数组中的所有元素复制到另一个一维数组中
Exists()	判断数组中是否包含给定的元素
GetLength()	获取 Array 的指定维中的元素数
GetLowerBound()	获取 Array 中指定维度的下限
GetUpperBound()	获取 Array 中指定维度的上限
GetValue()	获取 Array 中指定位置的值
Reverse()	反转一维数组中元素的顺序
SetValue()	设置数组中指定位置的元素
Sort()	对一维数组元素按指定方式进行排序

代码 5.4 演示了 Array 类的一些方法的用法。

代码 5.4　Array 类的一些方法的用法示例代码

```
01    using System;
02    namespace ArrayApplication
03    {
04        class MyArray
05        {
06            static void Main(string[] args)
07            {
08                int[] list = { 34, 72, 13, 44, 25, 30, 10 };
09                Console.Write("原始数组： ");
10                foreach (int i in list)//遍历数组
11                {
12                    Console.Write(i + " ");
13                }
14                Console.WriteLine();
15                // 使用 Reverse()方法逆转数组
16                Array.Reverse(list);
17                Console.Write("逆转数组： ");
18                foreach (int i in list)
```

```
19              {
20                  Console.Write(i + " ");
21              }
22              Console.WriteLine();
23              // 使用Sort ()方法排序数组
24              Array.Sort(list);
25              Console.Write("排序数组: ");
26              foreach (int i in list)
27              {
28                  Console.Write(i + " ");
29              }
30              Console.WriteLine();
31             Console.ReadKey();
32          }
33      }
34   }
```

5.5.2 数组的遍历操作

foreach 语句提供一种简单、明了的方法来循环访问数组的元素。对于单维数组，foreach 语句以递增索引顺序处理元素（从索引 0 开始并以索引 Length−1 结束）。foreach 语句的语法格式如下：

```
foreach(数据类型  变量名 in 数组名)
{
    循环体语句。
}
```

例如有一个数组，可以使用 foreach 循环将其内的所有值依次显示出来。示例代码如代码 5.5 所示。

代码 5.5　使用 foreach 循环显示数组中的元素示例代码

```
01  int[] numbers = { 4, 5, 6, 1, 2, 3, -2, -1, 0 };
02  foreach (int i in numbers)
03  {
04      System.Console.Write("{0} ", i);
05  }
06  // Output: 4 5 6 1 2 3 -2 -1 0
```

扩展资源：实例 5.3-对已有数组进行逆转显示和排序显示实例。完整的代码及代码解析请扫描 5.5 节节首的二维码观看微课视频进行学习。

需要注意的是，使用 foreach 遍历数组元素时，不能直接对 foreach 中定义的临时变量进行值的修改。因为该临时变量不是原始数组中的一员，不能达到修改原始数组的目的。

微课视频 5-6

5.6 字符串

在第 3 章中，我们已经初步地认识了什么是字符串。字符串是所有编程语言在项目开发过程中涉及最多的部分。大部分项目的运行结果都需要以字符串的形式展示给客户，比如财务系统的报表、电子游戏的比赛结果、火车站的列车时刻表，等等。这些都需要经过