## c#在线指南网址: <https://wizardforcel.gitbooks.io/msdn-csharp/content/guide/13.html>

# c#中的数组,继承自抽象类Array

## 1.c#中的数组其实是一共对象,这个和c/c++里面数组是一段连续的内存空间不一样

|  |
| --- |
| int[] nums = { 1, 2, 3, 4, 5 }; |

## 2.获取数组是长度使用数组对象的length属性

|  |
| --- |
| int len = nums.length; |

## 3.可以使用Rank属性来获取数组是维数

|  |
| --- |
| int dimemsion = nums.Rank; |

### 实例

|  |
| --- |
| using System;  public class Example  {  public static void Main()  {  int[] array1 = new int[10];  int[,] array2= new int[10,3];  int[][] array3 = new int[10][];  Console.WriteLine("{0}: {1} dimension(s)",  array1.ToString(), array1.Rank);  Console.WriteLine("{0}: {1} dimension(s)",  array2.ToString(), array2.Rank);  Console.WriteLine("{0}: {1} dimension(s)",  array3.ToString(), array3.Rank);  }  }  // The example displays the following output:  // System.Int32[]: 1 dimension(s)  // System.Int32[,]: 2 dimension(s)  // System.Int32[][]: 1 dimension(s) |

## 4.c#创建数组

### 1.一维数组.

#### int[] nums = new int[] {10,20,30…},这种写法可以简写为int[] nums = {10,20,30…},注意如果是先声明,后赋值,就不能够简写

#### 下面的写法是错误的

|  |
| --- |
|  |

### 2.二维数组

#### 下列声明创建一个四行两列的二维数组。在[]里面有n给逗号,就是n+1维,如有一个就是2维

int[,] array = new int[4, 2];//声明二维数组

|  |
| --- |
| int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };  // The same array with dimensions specified.  int[,] array2Da = new int[4, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };  // A similar array with string elements.  string[,] array2Db = new string[3, 2] { { "one", "two" }, { "three", "four" },  { "five", "six" } }; |
| int[,] array4 = { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } }; |

### 3.三维数组

#### 下列声明创建一个三维（4、2 和 3）数组。

int[, ,] array1 = new int[4, 2, 3];

|  |
| --- |
| int[, ,] array3D = new int[,,] { { { 1, 2, 3 }, { 4, 5, 6 } },  { { 7, 8, 9 }, { 10, 11, 12 } } };  // The same array with dimensions specified.  int[, ,] array3Da = new int[2, 2, 3] { { { 1, 2, 3 }, { 4, 5, 6 } },  { { 7, 8, 9 }, { 10, 11, 12 } } }; |

## 5.交错数组

交错数组是元素为数组的数组。交错数组元素的维度和大小可以不同。交错数组有时称为“数组的数组”。以下示例说明如何声明、初始化和访问交错数组。

下面声明一个由三个元素组成的一维数组，其中每个元素都是一个一维整数数组：

int[][] jaggedArray = new int[3][];

必须初始化 jaggedArray 的元素后才可以使用它。可以如下例所示初始化该元素：

jaggedArray[0] = new int[5];

jaggedArray[1] = new int[4];

jaggedArray[2] = new int[2];

每个元素都是一个一维整数数组。第一个元素是由 5 个整数组成的数组，第二个是由 4 个整数组成的数组，而第三个是由 2 个整数组成的数组。

也可以使用初始值设定项用值填充数组元素，在这种情况下不需要数组大小。例如：

jaggedArray[0] = new int[] { 1, 3, 5, 7, 9 };

jaggedArray[1] = new int[] { 0, 2, 4, 6 };

jaggedArray[2] = new int[] { 11, 22 };

还可以在声明数组时将其初始化，如：

int[][] jaggedArray2 = new int[][]

{

new int[] {1,3,5,7,9},

new int[] {0,2,4,6},

new int[] {11,22}

};

可以使用下面的速记格式。请注意：不能从元素初始化中省略 **new** 运算符，因为不存在元素的默认初始化：

int[][] jaggedArray3 =

{

new int[] {1,3,5,7,9},

new int[] {0,2,4,6},

new int[] {11,22}

};

交错数组是数组的数组，因此其元素是引用类型并初始化为 **null**。

可以如下例所示访问个别数组元素：

// Assign 77 to the second element ([1]) of the first array ([0]):

jaggedArray3[0][1] = 77;

// Assign 88 to the second element ([1]) of the third array ([2]):

jaggedArray3[2][1] = 88;

可以混合使用交错数组和多维数组。下面声明和初始化一个一维交错数组，该数组包含大小不同的三个二维数组元素。有关二维数组的详细信息，请参阅[多维数组（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/2yd9wwz4.aspx)。

int[][,] jaggedArray4 = new int[3][,]

{

new int[,] { {1,3}, {5,7} },

new int[,] { {0,2}, {4,6}, {8,10} },

new int[,] { {11,22}, {99,88}, {0,9} }

};

可以如本例所示访问个别元素，该示例显示第一个数组的元素 [1,0] 的值（值为 5）：

System.Console.Write("{0}", jaggedArray4[0][1, 0]);

方法 Length 返回包含在交错数组中的数组的数目。例如，假定您已声明了前一个数组，则此行：

System.Console.WriteLine(jaggedArray4.Length);

返回值 3。

本例生成一个数组，该数组的元素为数组自身。每一个数组元素都有不同的大小。

class ArrayTest

{

static void Main()

{

// Declare the array of two elements:

int[][] arr = new int[2][];

// Initialize the elements:

arr[0] = new int[5] { 1, 3, 5, 7, 9 };

arr[1] = new int[4] { 2, 4, 6, 8 };

// Display the array elements:

for (int i = 0; i < arr.Length; i++)

{

System.Console.Write("Element({0}): ", i);

for (int j = 0; j < arr[i].Length; j++)

{

System.Console.Write("{0}{1}", arr[i][j], j == (arr[i].Length - 1) ? "" : " ");

}

System.Console.WriteLine();

}

// Keep the console window open in debug mode.

System.Console.WriteLine("Press any key to exit.");

System.Console.ReadKey();

}

}

/\* Output:

Element(0): 1 3 5 7 9

Element(1): 2 4 6 8

\*/

## 6.数组作为函数参数

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace cSharpbasic  {  internal class Program  {    static void printArray(string[] arr)  {  foreach (var item in arr)  {  Console.Write("{0} ", item);  }  }  static string[] reverseArr(string[] arr)  {  return arr.Reverse().ToArray();  }  static void Main(string[] args)  {  Console.WriteLine("Before Reverse\n");  printArray(new string[] { "one", "two", "three" });  Console.WriteLine("\n");  string[] arr = reverseArr(new string[] { "one", "two", "three" });  Console.WriteLine("After Reverse\n");  printArray(arr);  }  }  } |
|  |

## 7. 使用 ref 和 out 传递数组参数

### out关键字说明这个数组是用来接收值的,你需要在函数里面给他赋值,ref就是参数有值但是你可以修改他的值

### 实例

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace cSharpbasic  {  internal class Program  {    static void FillArrOut(out int[] arr)  {  arr = new int[] { 100, 200, 300 };//使用out的参数不需要先初始化,需要在函数里面初始化  }  static void testFillArrOut()  {  int[] ints;  FillArrOut(out ints);  foreach (var item in ints)  {  Console.Write("{0},", item);  }  }  static void FillArrRef(ref int[] arr)  {  //arr = new int[] { 100, 200, 300 };  arr[arr.Length-1] = 1000;  }  static void testFillArrRef()  {  int[] ints = {1,2,3};  FillArrRef(ref ints);//使用ref的参数必须要先初始化,否则报错  foreach (var item in ints)  {  Console.Write("{0},", item);  }  }  static void Main(string[] args)  {    //testFillArrOut();  testFillArrRef();  }  }  } |

## 8. 隐式类型的数组（C# 编程指南）

可以创建隐式类型的数组，在这样的数组中，数组实例的类型是从数组初始值设定项中指定的元素推断而来的。有关任何隐式类型变量的规则也适用于隐式类型的数组。有关更多信息，请参见[隐式类型的局部变量（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/bb384061.aspx)。

在查询表达式中，隐式类型的数组通常与匿名类型以及对象初始值设定项和集合初始值设定项一起使用。

下面的示例演示如何创建隐式类型的数组：

class ImplicitlyTypedArraySample

{

static void Main()

{

var a = new[] { 1, 10, 100, 1000 }; // int[]

var b = new[] { "hello", null, "world" }; // string[]

// single-dimension jagged array

var c = new[]

{

new[]{1,2,3,4},

new[]{5,6,7,8}

};

// jagged array of strings

var d = new[]

{

new[]{"Luca", "Mads", "Luke", "Dinesh"},

new[]{"Karen", "Suma", "Frances"}

};

}

}

请注意，在上一个示例中，没有在初始化语句的左侧对隐式类型的数组使用方括号。另请注意，交错数组就像一维数组那样使用 **new []** 进行初始化。

### 对象初始值设定项中的隐式类型的数组

创建包含数组的匿名类型时，必须在该类型的对象初始值设定项中对数组进行隐式类型化。在下面的示例中，contacts 是一个隐式类型的匿名类型数组，其中每个匿名类型都包含一个名为 PhoneNumbers 的数组。请注意，对象初始值设定项内部未使用 **var** 关键字。

var contacts = new[]

{

new {

Name = " Eugene Zabokritski",

PhoneNumbers = new[] { "206-555-0108", "425-555-0001" }

},

new {

Name = " Hanying Feng",

PhoneNumbers = new[] { "650-555-0199" }

}

};

# 参考: [C#基础：ref和out的区别](https://www.cnblogs.com/gjahead/archive/2008/02/28/1084871.html)

ref或out关键字。ref和out这两个关键字都能够提供相似的功效，其作用也很像C中的指针变量。它们的区别是：

1、使用ref型参数时，传入的参数必须先被初始化。对out而言，必须在方法中对其完成初始化。

2、使用ref和out时，在方法的参数和执行方法时，都要加Ref或Out关键字。以满足匹配。

3、out适合用在需要retrun多个返回值的地方，而ref则用在需要被调用的方法修改调用者的引用的时候。

注：在C#中，方法的参数传递有四种类型：传值（by value），传址（by reference），输出参数（by output），数组参数（by array）。传值参数无需额外的修饰符，传址参数需要修饰符ref，输出参数需要修饰符out，数组参数需要修饰符params。传值参数在方法调用过程中如果改变了参数的值，那么传入方法的参数在方法调用完成以后并不因此而改变，而是保留原来传入时的值。传址参数恰恰相反，如果方法调用过程改变了参数的值，那么传入方法的参数在调用完成以后也随之改变。实际上从名称上我们可以清楚地看出两者的含义--传值参数传递的是调用参数的一份拷贝，而传址参数传递的是调用参数的内存地址，该参数在方法内外指向的是同一个存储位置。

方法参数上的 ref 方法参数关键字使方法引用传递到方法的同一个变量。当控制传递回调用方法时，在方法中对参数所做的任何更改都将反映在该变量中。

若要使用 ref 参数，必须将参数作为 ref 参数显式传递到方法。ref 参数的值被传递到 ref 参数。

传递到 ref 参数的参数必须最先初始化。如果不初始化,就会报错

|  |
| --- |
|  |

将此方法与 out 参数相比，后者的参数在传递到 out 参数之前不必显式初始化。

属性不是变量，不能作为 ref 参数传递。

如果两种方法的声明仅在它们对 ref 的使用方面不同，则将出现重载。但是，无法定义仅在 ref 和 out 方面不同的重载。

out

方法参数上的 out 方法参数关键字使方法引用传递到方法的同一个变量。当控制传递回调用方法时，在方法中对参数所做的任何更改都将反映在该变量中。

当希望方法返回多个值时，声明 out 方法非常有用。使用 out 参数的方法仍然可以返回一个值。一个方法可以有一个以上的 out 参数。

若要使用 out 参数，必须将参数作为 out 参数显式传递到方法。out 参数的值不会传递到 out 参数。

不必初始化作为 out 参数传递的变量。然而，必须在方法返回之前为 out 参数赋值。

ref是有进有出，而out是只出不进。