MẢNG TRONG C#

Mảng là một cấu trúc dữ liệu đơn giản và có mặt trong hầu hết các ngôn ngữ lập trình, trong đó có C#. Dữ liệu kiểu mảng có rất nhiều ứng dụng và các thuật toán áp dụng cho nó. Trong bài học này chúng ta sẽ đi vào kiểu dữ liệu collection cơ bản nhất, đơn giản nhất và phổ biến nhất trong C# – cấu trúc mảng (array). Đây là bài học đầu tiên về các kiểu dữ liệu collection tuyến tính (linear collection) của C#.

Mảng một chiều các khái niệm và thao tác cơ bản

Mảng (array) là một cấu trúc dữ liệu phổ biến bậc nhất, có mặt trong hầu hết các ngôn ngữ lập trình, và C# không phải là ngoại lệ.

Mảng là một tập các dữ liệu **có cùng kiểu**. Mỗi dữ liệu đơn lẻ trong mảng được gọi là *phần tử* của mảng. Các phần tử được **đánh chỉ số** (index) và được sắp xếp liền kề nhau thành một chuỗi. Kiểu của phần tử được gọi là *kiểu cơ sở* của mảng. Kiểu cơ sở có thể là các kiểu dữ liệu sẵn có của C# hoặc kiểu do người dùng tự định nghĩa. Chỉ số của mảng trong C# **bắt đầu là 0**, nghĩa là phần tử đầu tiên có chỉ số 0, phần tử thứ hai có chỉ số là 1, v.v..

Khai báo biến mảng

Trong C#, mảng là object, và chúng đều được tạo ra từ lớp `System.Array`. Do đó, chúng ta có thể khai báo biến mảng như khai báo các object bình thường. Để tiện lợi và quen thuộc cho người lập trình, C# tạo ra một cấu trúc cú pháp riêng để **khai báo mảng**.

```
//cú pháp chung
kiểu_cơ_sở[] biến_mảng;
//ví dụ dưới đây khai báo biến mảng names với kiểu cơ sở là string
string[] names;
// string là kiểu cơ sở; names là tên biến mảng; cặp dấu [] là bắt buộc, nó báo rằng đây là khai
// khai báo mảng số nguyên
int[] numbers;
```

Lưu ý phân biệt các khái niệm *biến mảng*, *kiểu mảng*, *kiểu cơ sở*. Trong ví dụ trên, `string` là *kiểu cơ sở*, còn `string[]` được gọi là *kiểu mảng*, còn names là *biến mảng*. Trong C# viết `string[]` có nghĩa là đây là một kiểu mảng, trong đó kiểu cơ sở của phần tử là `string`, `string[] names` là lệnh khai báo biến mảng.

Một biến mảng khi được khai báo sẽ chưa được cấp phát bộ nhớ ngay. Nói cách khác, nếu chỉ khai báo thì chưa thể sử dụng được biến mảng. Để sử dụng biến mảng chúng ta phải **khởi tạo mảng**. Điều này cũng tương tự như đối với các object bình thường khác trong C#. Mỗi object phải được khởi tạo sau khi khai báo và trước khi sử dụng.

Khởi tạo biến mảng

Khởi tạo biến là việc yêu cầu cấp phát bộ nhớ cho biến đó. Chỉ khi khởi tạo xong chúng ta mới có thể sử dụng được biến mảng. C# cung cấp một số cách khác nhau để khởi tạo biến mảng.

Cách thứ nhất là cung cấp số lượng phần tử cần dùng.

```
// khởi tạo mảng string với 10 phần tử
string[] names = new string[10];
```

Cách này tạo ra một mảng string có 10 phần tử. Mỗi phần tử sẽ nhận giá trị mặc định là null.

Trong C#, các biến khi khai báo mà không gán sẵn giá trị sẽ đều nhận một *giá trị mặc định* (default value). Giá trị này phụ thuộc vào kiểu của nó. Ví dụ biến kiểu int có giá trị mặc định là 0, kiểu bool là false, object là null. Như vậy, ở ví dụ trên, mỗi phần tử của mảng names có giá trị mặc định là null (do string là object).

Cách thứ hai là cung cấp sẵn danh sách phần tử cho biến mảng.

```
int[] numbers = new int[] {1, 2, 3, 4, 5};
```

Cách thứ ba là một cấu trúc tắt có tên gọi là **array initializer**. Trong cấu trúc này bạn không sử dụng được từ khóa var mà bắt buộc phải chỉ định rõ kiểu mảng.

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

Lưu ý rằng, một khi được khởi tạo, số lượng phần tử của mảng sẽ không thể thay đổi được.

Truy xuất phần tử của mảng

Một khi mảng đã được khởi tạo, chúng ta có thể sử dụng nó để lưu trữ dữ liệu. Việc sử dụng bao gồm hai phần: truy xuất phần tử, truy xuất thông tin meta.

Các phần tử của mảng trong C# được truy xuất (đọc/ghi) trực tiếp qua chỉ số với cú pháp như sau:

```
// gán giá trị cho phần tử dùng phép toán indexing
biến_mảng[chỉ_số] = giá_trị;
// hoặc dùng phương thức SetValue theo kiểu hướng đối tượng
biến_mảng.SetValue(giá_trị, chỉ_số);
// đọc giá trị bằng phép toán indexing
biến = biến_mảng[chỉ_số];
// hoặc dùng phương thức GetValue
biến = biến_mảng.GetValue(chỉ_số);
```

Ví dụ

```
//ví dụ
names[0] = "Donald Trump";
names.SetValue("Donald Trump", 0);
var current_us_president = names[0];
var next_us_president = names.GetValue(0);
```

Chỉ số có giá trị **từ 0 đến n-1**, với *n* là *số phần tử* của mảng. Nếu chỉ số nằm ngoài dải này, việc truy xuất sẽ báo lỗi `IndexOutOfRangeException`.

Đọc metadata của mảng

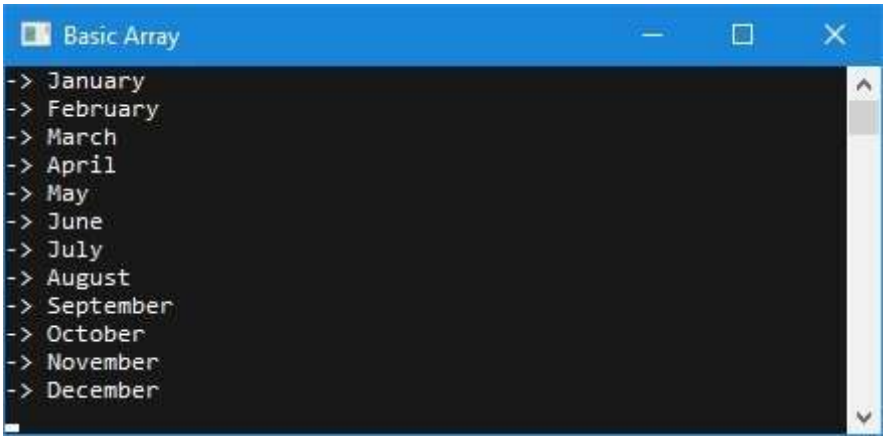
Metadata là những thông tin về bản thân mảng, như số lượng phần tử, kiểu cơ sở, v.v.. Do mảng đều là các object thuộc kiểu System.Data, chúng ta có thể sử dụng các thuộc tính (và phương thức) của lớp này để đọc metadata của mảng.

- Thuộc tính `Length` / `LongLength` (read-only): số phần tử của mảng
- Phương thức `GetLength` / `GetLongLength`: đọc số phần tử của mảng
- Thuộc tính `Rank` (read-only): số chiều của mảng. Chúng ta sẽ làm quen với mảng nhiều chiều ngay sau đây.
- Phương thức `GetType`: lấy thông tin về kiểu của mảng.

Một ví dụ tổng hợp về sử dụng mảng cơ bản

Tự thực hiện lại ví dụ dưới đây để củng cố các kỹ thuật làm việc cơ bản với mảng. Trong ví dụ này chúng ta sẽ lưu trữ tên các tháng (trong tiếng Anh) vào một mảng và in ra màn hình console.

```
using System;
using System.Globalization;
namespace P01_SingleDimension
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Title = "Basic Array";
            // khai báo và khởi tạo mảng chứa tên 12 tháng trong tiếng Anh
            string[] months = new string[12];
            // duyệt qua các phần tử và gán giá trị
            for (int month = 1; month <= 12; month++)
            {
                DateTime firstDay = new DateTime(DateTime.Now.Year, month, 1);
                string name = firstDay.ToString("MMMM", CultureInfo.CreateSpecificCulture("en"));
                months[month - 1] = name;
            }
            // duyệt qua các phần tử và in giá trị ra console
            foreach (string month in months)
            {
                Console.WriteLine($"-> {month}");
            }
            Console.ReadLine();
        }
    }
}
```



Chương trình mảng cơ bản trong c#

Kết quả chạy chương trình

Trong ví dụ trên bạn thấy một cấu trúc lặp đặc biệt: vòng lặp foreach. Nếu chưa biết về cấu trúc này, hãy đọc phần tiếp theo. Nếu bạn đã biết cấu trúc này thì có thể bỏ qua để chuyển thẳng sang nội dung về mảng nhiều chiều.

Duyệt mảng bằng foreach

C# cung cấp một cấu trúc điều khiển vòng lặp riêng để duyệt phần tử của các kiểu tập hợp như mảng: vòng lặp **foreach**.

Hãy cùng xem xét qua một ví dụ (bạn có thể thực hiện trên C# Interactive):

```
> int[] integers = new int[] { 2, 4, 6, 8, 10 };
. foreach(var i in integers)
. {
.     Console.Write($"{i}\t");
. }
2   4   6   8   10
>
```

Cấu trúc lặp này chỉ áp dụng được với các kiểu tập hợp. Nói chính xác hơn, foreach chỉ áp dụng được với các class thực thi giao diện IEnumerable hoặc IEnumerable.

Cú pháp chung của vòng lặp foreach như sau:

```
foreach(data_type var_name in collection_variable)
{
    // statements to be executed
}
```

Vòng lặp foreach tự động duyệt qua các phần tử của mảng *collection_variable*. Khi duyệt đến phần tử nào, giá trị của phần tử đó sẽ được copy vào biến *var_name*. Bên trong vòng lặp, bạn trực tiếp sử dụng luôn biến *var_name* thay cho dùng chỉ số và phép toán index.

Quay lại ví dụ ở trên. Vòng lặp foreach sẽ tự động duyệt qua mảng integers. Khi nó dừng ở phần tử nào, giá trị của phần tử đó sẽ được copy sang biến tạm i. Trong thân vòng lặp, bạn sử dụng giá trị của phần tử đó thông qua biến tạm i (ở đây chỉ đơn giản là in ra console).

Có một số lưu ý sau khi sử dụng foreach:

foreach là cấu trúc duyệt mảng (và các collection) an toàn nhất và đơn giản nhất trong C#. Nếu mảng không có phần tử nào, foreach đơn giản là không thực hiện. Không cần phải kiểm tra chỉ số như các vòng lặp khác.

Foreach duyệt phần tử của mảng luôn theo một hướng duy nhất từ đầu đến cuối mảng, không thể theo chiều ngược lại.

Do biến tạm var_name chỉ chứa bản copy của phần tử tập hợp, bạn không thể thay đổi được giá trị của phần tử tập hợp khi tác động vào biến var_name. Nói cách khác, foreach là vòng lặp chỉ đọc để duyệt qua các phần tử của mảng.

Nếu có lệnh thay đổi giá trị của biến tạm var_name, bạn sẽ gặp lỗi. Xem ví dụ sau:

```
> int[] integers = new int[] { 2, 4, 6, 8, 10 };
. foreach(var i in integers)
. {
.     i *= 2;
.     Console.Write($"{i}\t");
. }
(4,5): error CS1656: Cannot assign to 'i' because it is a 'foreach iteration variable'
>
```

Ở đây biểu thức gán i*= 2 là chỗ gây lỗi. Bạn không thể thay đổi giá trí của biến tạm i, do nó là bản copy chỉ đọc của phần tử của mảng.

Mảng nhiều chiều

Mảng chúng ta xem xét ở phần trên có tên gọi là *mảng một chiều*. Chúng ta hình dung nó như một danh sách (chuỗi) các phần tử. Mảng trong C# có thể có nhiều hơn một chiều, gọi chung là *mảng nhiều/đa chiều*.

Ví dụ, chúng ta có thể có *mảng hai chiều*. Khi đó ta có thể hình dung nó như một **bảng (chữ nhật)**, trong đó mỗi ô là một phần tử. Với *mảng ba chiều*, chúng ta hình dung nó như một **khối hộp** lớn, gồm nhiều hộp nhỏ (giống như khối rubic). Mỗi hộp nhỏ là một phần tử.

C# hỗ trợ mảng có tối đa 32 chiều. Chắc rất ít khi phải dùng đến loại mảng nhiều chiều như vậy! Các mảng lớn hơn 3 chiều thường rất khó hình dung.

Chúng ta xem xét cách làm việc với mảng đa chiều qua hai trường hợp cụ thể: mảng hai chiều, mảng ba chiều. Mảng nhiều chiều hơn cũng tuân theo quy tắc tương tự.

Mảng hai chiều

```
// khai báo và khởi tạo một mảng hai chiều (3x4) của các số nguyên
int[,] numbers = new int[3, 4]
{
    {11, 12, 13, 14 },
    {21, 22, 23, 24 },
    {31, 32, 33, 34 }
};
```

Trong ví dụ trên `int[,]` là kiểu mảng hai chiều trong đó các phần tử thuộc kiểu `int`, `numbers` là biến mảng, kích thước là 3 hàng x 4 cột. Có thể dễ dàng nhận ra cú pháp khai báo mảng đa chiều: với mỗi một chiều bổ sung, chúng ta thêm một dấu phẩy vào giữa cặp dấu `[]`.

Lưu ý cách chúng ta khởi tạo giá trị. Có thể hình dung mảng hai chiều tổ chức dữ liệu theo hàng và cột. Do đó, chúng ta khởi tạo mảng bằng cách:

1. tạo ra các hàng, các phần tử của mỗi hàng đặt trong cặp `{}` và phân tách bằng dấu phẩy như một mảng đơn chiều;
2. ghép các hàng lại với nhau (phân tách bằng dấu phẩy) và đặt tiếp trong cặp `{}`.

Như vậy, mảng 2 chiều là sự mở rộng của mảng 1 chiều trong mặt phẳng.

Dấu phẩy cũng có tác dụng ngăn cách các chỉ số khi truy cập vào phần tử của mảng đa chiều. Mảng hai chiều phát sinh thêm một dấu phẩy để có thể ghi hai chỉ số. Do mảng hai chiều hình dung như một bảng, nó sẽ bao gồm hai chỉ số: chỉ số hàng (đứng trước dấu phẩy) và chỉ số cột (đứng sau dấu phẩy).

```
numbers[0, 0] = 11; // truy xuất phần tử đầu tiên (hàng thứ nhất - chỉ số 0, cột thứ nhất - chỉ s
var number34 = numbers[2, 3] = 34; // truy xuất phần tử hàng thứ 3 (chỉ số 2) và cột thứ 4 (chỉ s
```

Khi làm việc với mảng nhiều chiều nên chú ý phân biệt *kích thước* của mảng với *chỉ số* phần tử của mảng.

Mảng ba chiều

Tương tự, mảng ba chiều phải dùng hai dấu phẩy để có thể ghi 3 kích thước.

```
int[,,] numbers2 = new int[2, 3, 4]
{
    {
        {111, 112, 113, 114 },
        {121, 122, 123, 124 },
        {131, 132, 133, 134 }
    },
    {
        {211, 212, 213, 214 },
        {221, 222, 223, 224 },
        {231, 232, 233, 234 }
    },
};
```

Chúng ta có thể hình dung mảng 3 chiều chính là các mảng 2 chiều xếp lớp chồng lên nhau (trong không gian). Hai chỉ số cuối chính là số lượng hàng và cột của bảng 2 chiều. Chỉ số đầu tiên chính là số lượng lớp.

Việc truy xuất phần tử của mảng 3 chiều cũng theo quy tắc tương tự:

```
numbers2[0, 0, 0] = 111; // truy xuất phần tử đầu tiên
var number234 = numbers2[1, 2, 3]; // truy xuất phần tử cuối cùng (giá trị 234)
```

Theo quy tắc trên chúng ta hoàn toàn có thể khai báo và khởi tạo các mảng có số chiều lớn hơn nữa.

Một ví dụ về sử dụng mảng đa chiều

Trong ví dụ dưới đây chúng ta sẽ tạo và in ra console bảng cửu chương sử dụng mảng hai chiều.

```
using System;
namespace P02_MultiDimension
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Title = "Multiplication table";
            // khai báo và khởi tạo mảng hai chiều chứa bảng cửu chương
            var multiplications = new int[10, 10];
            // gán giá trị cho các phần tử của bảng cửu chương
            for (int r = 0; r < multiplications.GetLength(0); r++)
            {
                for (int c = 0; c < multiplications.GetLength(1); c++)
                {
                    multiplications[r, c] = (r + 1) * (c + 1);
                }
            }
            // in ra màn hình
            for (int c = 0; c <= multiplications.GetLength(1); c++)
            {
                Console.ForegroundColor = ConsoleColor.Cyan;
                if (c == 0) Console.Write("{0, 4}", "");
                else
                    Console.Write("{0, 4}", c);
            }
            Console.WriteLine();
            for (int r = 0; r < multiplications.GetLength(0); r++)
            {
                Console.ForegroundColor = ConsoleColor.Yellow;
                Console.Write("{0, 4}", r + 1);
                Console.ResetColor();
                for (int c = 0; c < multiplications.GetLength(1); c++)
                {
                    Console.Write("{0, 4}", multiplications[r, c]);
                }
```

```
                Console.WriteLine();
            }
            Console.ReadKey();
        }
    }
}
```

Chương trình mảng 2 chiều C#

Kết quả chạy chương trình

Trong ví dụ trên lưu ý phương thức `GetLength`. Phương thức này cho phép lấy thông tin về số lượng phần tử trong từng chiều của mảng. Nếu mảng một chiều thì giá trị tham số luôn là 0. Nếu là mảng hai chiều thì giá trị tham số có thể là 1 (số hàng) hoặc 0 (số cột).

Mảng răng cưa

Mảng răng cưa (Jagged Array) là một loại mảng đặc biệt trong C# và cũng thường được gọi là *mảng của mảng*. Có thể hình dung mảng răng cưa là một mảng một chiều, trong đó **mỗi phần tử của nó lại là một mảng**, thay vì là một dữ liệu cụ thể.

Mỗi mảng phần tử có thể có kích thước khác nhau nhưng bắt buộc phải có chung kiểu cơ sở. Mảng phần tử thậm chí có thể không cần khởi tạo.

Nếu bạn vẫn chưa hình dung được sự khác biệt giữa mảng răng cưa và mảng hai chiều:

- Mảng hai chiều bạn hãy hình dung nó như một ma trận. Giả sử mảng 3×4, tức là có 3 hàng x 4 cột thì mỗi hàng đều phải có đủ 4 cột.
- Mảng răng cưa thì khác. Giả sử cũng có 3 hàng, nhưng hàng thứ nhất có thể có 5 cột, hàng thứ hai chỉ có 2 cột, hàng thứ 3 lại có 4 cột. Rõ ràng nó "cộc lệch" chứ không tạo thành dạng ma trận.

Hãy cùng xem ví dụ sau để hiểu cách sử dụng của mảng răng cưa.

```
int[][] numbers = new int[5][];
numbers[0] = new int[] { 1, 2, 3 };
```

```
numbers[2] = new int[] { 1, 2, 3, 4, 5 };
numbers[4] = new int[] { 1, 2 };
```

Trong ví dụ trên chúng ta đã khai báo và khởi tạo biến numbers là một mảng có thể chứa đến 5 mảng thành viên. Mỗi thành viên này là một mảng một chiều và được khởi tạo riêng rẽ theo cú pháp chúng ta đã biết.

Truy xuất phần tử của mảng răng cưa có chút khác biệt so với mảng thông thường:

```
var value1 = numbers[0][0]; // truy xuất giá trị đầu tiên của mảng thành viên đầu tiên
var value2 = numbers[2][4]; // truy xuất giá trị cuối cùng của mảng thành viên số 2
```

Chúng ta có thể nhận thấy, việc truy xuất giá trị của mảng răng cưa bao gồm hai thông tin: chỉ số của mảng thành viên trong cặp [] thứ nhất; chỉ số của phần tử trong mảng thành viên trong cặp [] thứ hai.

Cùng xem một ví dụ khác.

```
int[][,] numbers = new int[10][,];
numbers[0] = new int[2, 3];
numbers[1] = new int[3, 4];
numbers[2] = new int[,] { { 1, 3, 5 }, { 2, 4, 6 } };
```

Ở đây chúng ta lại khai báo mảng numbers là một mảng của các mảng hai chiều. Nghĩa là, mỗi phần tử của nó là một mảng hai chiều mà chúng ta phải tự khởi tạo trước khi dùng.

Việc truy xuất phần tử của mảng này cũng theo quy tắc ở trên: chọn chỉ số của mảng thành viên, chọn chỉ số của phần tử cụ thể trong mảng thành viên.

```
var value = numbers[2][2,2]; // truy xuất phần tử ở vị trí 2,2 của mảng thành viên số 2
```