KIỂU DỮ LIỆU

Kiểu dữ liệu (data type, hay đơn giản là type) trong C# (cũng như các ngôn ngữ khác) là một đặc tính của dữ liệu nhằm thông báo cho C# compiler biết về ý định sử dụng dữ liệu của lập trình viên. Một trong những việc đầu tiên cần sử dụng đến kiểu dữ liệu là khai báo biến và hằng mà chúng ta đã biết. C# nghiêm ngặt hơn nhiều so với các ngôn ngữ khác về vấn đề kiểu dữ liệu. Ngoài ra có nhiều điều khác biệt về kiểu dữ liệu của C# mà bạn không thể không biết.

Kiểu dữ liệu của C# vs .NET

Trong C# bạn có thể sử dụng đến hàng chục ngàn kiểu dữ liệu khác nhau! Thật vậy. Đó là những kiểu dữ liệu được định nghĩa trong hàng loạt thư viện của .NET Framework. Tuy vậy, nếu nói một cách nghiêm ngặt thì C# lại không hề định nghĩa kiểu dữ liệu nào! Đây là một điều rất lạ, rất khác biệt của C#.

Vấn đề là, C# không tồn tại độc lập. Nó là một ngôn ngữ gắn liền với .NET. .NET mới là người cung cấp hàng chục nghìn kiểu dữ liệu cho C#. .NET không chỉ cung cấp những kiểu dữ liệu "đỉnh cao" mà nó cung cấp cả những kiểu dữ liệu cơ bản nhất mà nhẽ ra ngôn ngữ thường tự định nghĩa như số nguyên, số thực, logic, v.v..

Để đơn giản hóa code, C# định nghĩa các **biệt danh** (alias) riêng cho một số kiểu cơ bản của .NET bằng từ khóa. Ví dụ, `int` (C#) là biệt danh của `System.Int32` (.NET), `string` (C#) là biệt danh của `System.String` (.NET). Các biệt danh này làm cho C# nhìn rất giống C/C++ hay Java nhưng bản chất lại khác nhau.

Ngoài việc tạo biệt danh, C# đơn giản hóa cú pháp cho việc sử dụng chúng.

Đây là lý do khiến nhiều bạn khi làm việc với C# thắc mắc sự khác biệt giữa các tên kiểu, một số toàn viết thường (double, bool, string, char) với một số viết hoa (Double, Boolean, String, Char). Bản chất chúng nó là một nhưng cách sử dụng khác nhau một chút. C# khuyến khích sử dụng các biệt danh (nếu có) để code nhìn bớt phức tạp.

Dưới đây là danh sách các kiểu dữ liệu cơ bản (alias) của C# và kiểu tương ứng của .NET. Chúng ta sẽ đi sâu vào từng nhóm sau.

- Các kiểu số nguyên
- Các kiểu số thực
- Kiểu decimal của
- Kiểu logic bool
- Kiểu ký tự
- Kiểu object và string

Các bảng trên có mục đích giúp bạn có cảm nhận chung về tương quan giữa kiểu dữ liệu của C# và kiểu tương ứng của .NET. Bạn không cần ghi nhớ hay học thuộc chúng. Chúng ta sẽ xem xét chi tiết từng kiểu dữ liệu sau.

Đặc điểm của các kiểu dữ liệu cơ bản

Dưới đây chúng ta sẽ giới thiệu qua một số đặc điểm của các kiểu dữ liệu cơ bản của C#. Mục này hướng tới các bạn đã có nền tảng ở một ngôn ngữ lập trình khác để giúp bạn nhanh chóng nhìn thấy sự khác biệt của C#. Các bạn có xuất phát điểm là C/C++ hay Java có thể dễ dàng và nhanh chóng tiếp cận các kiểu cơ sở này.

Các kiểu số nguyên

Như bạn đã thấy trong bảng trên, C# (và .NET) cung cấp 8 kiểu số nguyên, phân biệt ở số byte để biểu diễn và vùng giá trị. Tên của các kiểu này hoàn toàn giống như trong Java hay C++. Cách sử dụng cũng hoàn toàn tương tự. Tuy nhiên có những điểm khác biệt cần lưu ý.

int và byte

Kiểu **int** của C# luôn luôn chiếm **4 byte** (32 bit). Trong C++, số bit của kiểu int thay đổi phụ thuộc vào platform (vd, trên windows là 32 bit).

Kiểu **byte** là 8 bit, có dải giá trị từ **0 đến 255**, và **không** thể chuyển đổi qua lại với kiểu char như trong C. Kiểu byte luôn luôn không dấu (khác với C). Nếu muốn sử dụng số nguyên 8 bit có dấu, bạn phải dùng kiểu sbyte.

Cơ số

Tất cả các kiểu số nguyên đều có thể nhận giá trị biểu diễn ở nhiều **cơ số** (base) khác nhau: cơ số 10 (decimal), 16 (hex), 8 (octal), 2 (binary). Giá trị biểu diễn ở các cơ số khác 10 phải sử dụng thêm tiếp tố (**prefix**) tương ứng.

```
long x = 0x12ab; // số hexa, prefix là 0x hoặc 0X
byte y = 0b1100; // số nhị phân, prefix là 0b hoặc 0B
int z = 01234; // số hệ cơ số 8, prefix là 0
```

Digit separator

C# 7 cho phép sử dụng dấu `_` **giữa** các chữ số để tách các chữ số cho dễ đọc hơn với các giá trị lớn. Dấu `_` gọi là **digit separator**.

```
long l1 = 0x123_456_789_abc_def; // dấu _ giúp tách các chữ số cho dễ đọc
long l2 = 0x123456789abcdef; // cách viết thông thường
int bin = 0b1111_1110_1101; // viết tách các bit thế này dễ đọc hơn
```

Integer literal

Khi dùng từ khóa `var` để khai báo biến thuộc kiểu số nguyên, C# mặc định sẽ hiểu nó là kiểu `int`. Nếu muốn chỉ định giá trị nguyên thuộc một kiểu nào đó khác, bạn phải sử dụng một cách viết riêng gọi là **integer literal**.

Integer literal là các ký tự viết vào cuối giá trị số (**postfix**) để báo hiệu kiểu dữ liệu, bao gồm: `U` (hoặc `u`) báo hiệu số nguyên không dấu; `L` (hoặc `l`) báo hiệu giá trị thuộc kiểu long; `UL` (hoặc `ul`) cho kiểu ulong. Có thể sử dụng các ký tự này khi viết ở hệ cơ số khác 10. Ví dụ:

```
var i0 = 123; // c# mặc định coi đây là kiểu int
var i1 = 123u; // giá trị này thuộc kiểu uint
var i2 = 123l; // giá trị này thuộc kiểu long
var i3 = 123ul; // giá trị này thuộc kiểu ulong
var i4 = 0x123L; // giá trị kiểu long ở hệ hexa
```

Từ giờ về sau bạn sẽ còn gặp nhiều literal nữa. Literal (chính tả) là cách viết giá trị của từng kiểu dữ liệu.

Nếu bạn khai báo số nguyên có giá trị đủ lớn để thoát khỏi dải của int, C# sẽ tự chọn kiểu phù hợp có dải giá trị đủ bao trùm. Ví dụ:

```
var ui = 3000000000; // đây sẽ là kiểu uint
var l = 5000000000; // đây sẽ là kiểu long
```

Các kiểu số thực

C# chỉ cung cấp 2 loại số thực: `float` (`System.Single`) và `double` (`System.Double`). Các thông tin chi tiết bạn đã xem ở phần trên. `float` có dải địa chỉ nhỏ hơn và độ chính xác thấp hơn so với `double`.

Khi dùng từ khóa `var` với giá trị số thực, C# sẽ mặc định hiểu nó thuộc về kiểu double. Để chỉ định một giá trị thực thuộc kiểu `float`, bạn cần dùng postfix `F` (hoặc `f`) sau giá trị. `F` (hoặc `f`) được gọi là **float literal**.

```
var r1 = 1.234; // r1 thuộc kiểu double
var r2 = 1.234f; // r2 thuộc kiểu float
```

`decimal` (`System.Decimal`) là một dạng số thực đặc biệt chuyên dùng trong tính toán tài chính.

Literal cho decimal là `M` (hoặc `m`).

```
var d = 12.30M; // biến này thuộc kiểu decimal
```

Các kiểu số thực cũng hỗ trợ cách viết dạng khoa học (và có thể kết hợp với `float` `decimal`):

```
var d1 = 1.5E-20; // cách viết khoa học bình thường là 1,5*10^-20, kiểu double
var f1 = 1.5E-10F; // số 1,5*10^-10, kiểu float
var m1 = 1.5E-20M; // 1,5*10$-20, kiểu decimal
```

Kiểu luận lý

`Boolean` (.NET) hay `bool` (C#) chỉ nhận đúng hai giá trị: `true` và `false`. Đây cũng được gọi là **literal** của kiểu bool.

Trong C# không thể tự do chuyển đổi giữa bool và số nguyên như trong C/C++. Tức là bạn không thể sử dụng 0 thay cho false, giá trị khác 0 thay cho true như trong C. Biến khai báo thuộc kiểu bool chỉ có thể gán giá trị `true` hoặc `false`.

Kiểu ký tự

Kiểu `char` (C#) hay `System.Char` (.NET) dùng để biểu diễn ký tự đơn, mặc định là các ký tự Unicode 16 bit.

Character literal

Literal của kiểu char là cặp dấu nháy đơn. Ví dụ `'A'`, `'a'`, `'1'`, `'@'`.

```
var c = 'A';
```

Đừng nhầm lẫn với cặp dấu nháy kép – là literal của chuỗi ký tự. Nếu sử dụng lẫn lộn cặp nháy đơn và nháy kép, compiler sẽ báo lỗi hoặc hiểu sai ý định của bạn.

Bạn cũng có thể sử dụng mã Unicode của ký tự như sau: `'\u0041'`, `'\x0041'`.

```
var c1 = '\u0041';
var c2 = '\x0041';
```

Một cách khác nữa để biểu diễn ký tự là dùng mã decimal cùng với ép kiểu: `(char) 65`.

```
var c3 = (char) 65;
```

Escape sequence

Tương tự như C, C# cũng định nghĩa một số ký tự đặc biệt gọi là **escape sequence**:

- `\'`: dấu nháy đơn
- `\"`: dấu nháy kép
- `\\`: dấu backslash (dùng trong đường dẫn)
- `\0`: Null
- `\a`: cảnh báo (alert)
- `\b`: xóa lùi (backspace)
- `\n`: dòng mới
- `\r`: quay về đầu dòng
- `\t`: dấu tab ngang
- `\v`: dấu tab dọc

Kiểu chuỗi ký tự

Chuỗi (xâu) ký tự (`string` hoặc `System.String`), khác biệt với các kiểu dữ liệu bên trên, là một kiểu dữ liệu tham chiếu (reference type) . Trong khi các kiểu dữ liệu ở bên trên thuộc loại giá trị (value type). Sự khác biệt là gì bạn xem ở phần cuối bài.

Literal của string là cặp dấu ngoặc kép:

```
var str1 = "Hello world";
var emptyStr = ""; // đây là một chuỗi ký tự hợp lệ, gọi là xâu rỗng
```

Trong chuỗi ký tự có thể sử dụng ký tự escape sequence (bạn đã biết ở trên):

```
// chuỗi này chứa hai escape sequence \r và \n.
// nếu in ra console, con trỏ văn bản sẽ chuyển xuống đầu dòng tiếp theo
string message = "Press any key to continue\r\n";
Console.WriteLine(message);
// nếu in chuỗi này ra sẽ thu được x1 = 123     x2 = 456, tức là có 1 dấu tab ở giữa
string solutions = "x1 = 123\tx2 = 456";
Console.WriteLine(solutions);
```

Trong chuỗi không được có mặt ký tự `\` (backslash). Lý do là ký tự này được sử dụng trong escape sequence. Ví dụ, dưới đây là một chuỗi sai (bị báo lỗi cú pháp):

```
string path = "C:\Programs\Visual Studio"; // chuỗi này bị lỗi vì chứa ký tự \.
```

Nếu muốn viết ký tự `\` vào chuỗi, bạn phải viết nó hai lần:

```
string path = "C:\\Program\\Visual Studio"; // chuỗi này OK
```

hoặc thêm ký tự `@` vào đầu chuỗi. Ký tự `@` sẽ tắt chế độ diễn giải escape sequence.

```
string path = @"C:\Program\Visual Studio"; // chuỗi này OK vì ký tự @ sẽ tắt chế độ nhận diện esc
```

Trong chuỗi ký tự cũng có thể chứa biến và biểu thức. Các giá trị này được tính toán trước khi chèn vào đúng vị trí của nó trong chuỗi. Tính năng này có tên gọi là **string interpolation**. Interpolated string được bắt đầu bằng ký tự `$`.

```
int x1 = 123, x2 = 456;
string solution = $"x1 = {x1}     x2 = {x2}     x3 = {x1 + x2}"; // đây là một interpolated strin
Console.WriteLine(solution);
// nếu in ra console sẽ thu được x1 = 123     x2 = 456     x3 = 579
```

String interpolation là tính năng rất tiện lợi để tạo ra các xâu động từ biến và biểu thức.

Xâu là một loại dữ liệu đặc biệt và được sử dụng rất rộng rãi. Nội dung trong bài này chưa đủ để làm việc với xâu. Bài giảng này có bài học riêng về cách sử dụng xâu trong C#.

Kiểu object

`object` (`System.Object`) là kiểu dữ liệu đặc biệt trong C# và .NET. Nó là kiểu dữ liệu "tổ tiên" của mọi kiểu dữ liệu khác (root type). Đây cũng là một trong hai kiểu reference.

Bạn có thể hình dung như thế này. Trong các ngôn ngữ lập trình hướng đối tượng, các kiểu dữ liệu thường được tổ chức theo dạng phân cấp (hierarchy) như một cái cây. Trong đó kiểu dữ liệu cấp cao nhất, ở gốc của cây gọi là *root type*. Tất cả các kiểu còn lại đều là các nhánh xuất phát từ gốc. Nếu bạn hiểu khái niệm kế thừa thì object chính là tổ tông của tất cả các loại kiểu. Nói cách khác, mọi kiểu dữ liệu khác đều là con/cháu/chắt/chút/chít của object.

Chúng ta sẽ quay lại kiểu object khi học về class và kế thừa. Tạm thời bạn chỉ cần biết vậy là được.

Tuy nhiên có một phương thức quan trọng của object bạn cần biết: `ToString()`. Phương thức này có mặt trong mọi kiểu dữ liệu mà bạn đã biết (do cơ chế kế thừa từ `object`). Nó giúp chuyển đổi giá trị của kiểu tương ứng về chuỗi ký tự. Bạn sẽ thường xuyên cần đến nó khi viết giá trị của một biến ra console.

```
var a = 123.456; // a thuộc kiểu double
var strA = a.ToString(); // strA giờ là một chuỗi, có giá trị "123.456"
```

Phân loại kiểu dữ liệu

Stack và Heap

Để hiểu được cách thức phân loại kiểu dữ liệu, bạn cần nhớ lại một số vấn đề liên quan đến stack và heap.

Stack và heap đều là các vùng bộ nhớ trong RAM của máy tính nhưng được tổ chức và sử dụng cho các mục đích khác nhau.

Stack là vùng nhớ hoạt động theo mô hình LIFO (vào sau, ra trước) dùng để lưu trữ các biến tạm thời được tạo ra bởi các phương thức. Khi một biến được khai báo, nó được tự động đẩy vào stack (push).

Nếu phương thức kết thúc, tất cả các biến mà phương thức đó đã đẩy vào stack đều bị giải phóng (pop) và mất đi. Khi một biến bị đẩy khỏi stack, vùng nhớ nó đã chiếm có thể được sử dụng lại cho các biến khác.

Stack được CPU quản lý và tối ưu hóa cho nhiệm vụ lưu trữ biến cục bộ. Người lập trình không cần phải can thiệp vào vùng nhớ này. Tốc độ đọc ghi dữ liệu với stack rất cao, tuy nhiên kích thước của stack lại bị giới hạn.

Heap là một vùng nhớ khác cho phép chương trình tự do lưu trữ giá trị. Giá trị tạo và lưu trên heap không bị giới hạn về kích thước mà chỉ phụ thuộc và kích thước RAM. Tuy nhiên tốc độ đọc ghi dữ liệu trên heap chậm hơn so với stack.

Để sử dụng heap, chương trình phải tự mình xin cấp phát và giải phóng bộ nhớ chiếm dụng trên heap. Nếu không sử dụng hợp lý vùng nhớ này có thể dẫn đến rò bộ nhớ (memory leak), vốn rất phổ biến khi lập trình C/C++.

.NET framework hỗ trợ rất tốt việc cấp phát và quản lý bộ nhớ của chương trình qua GC (Garbage Collector) nên người lập trình .NET không cần để ý nhiều đến việc xin cấp phát và giải phóng bộ nhớ heap.

Kiểu giá trị và kiểu tham chiếu

Khi nói về hệ thống kiểu dữ liệu trong C# cần phân biệt hai nhóm: các kiểu giá trị (**value types**) và các kiểu tham chiếu (**reference types**). Cách phân loại này liên quan đến việc cấp và quản lý bộ nhớ cho biến.

Theo cách phân loại này, object và string thuộc nhóm reference, tất cả các kiểu còn lại thuộc nhóm value.

Sự phân biệt này rất quan trọng cho việc khởi tạo và gán giá trị của biến.

Sự khác biệt giữa hai nhóm này thể hiện ở nhiều vấn đề. Tuy nhiên, tạm thời hãy chấp nhận sự khác biệt cơ bản: kiểu giá trị lưu trữ dữ liệu trực tiếp trong biến; kiểu tham chiếu lưu trữ địa chỉ (tham chiếu) của dữ liệu (nằm ở nơi khác). Cụ thể hơn, dữ liệu của kiểu giá trị lưu trong *stack*, trong khi dữ liệu của kiểu tham chiếu (thường gọi là object) lưu tron *heap*. Bản thân biến thuộc kiểu tham chiếu chỉ chứa địa chỉ tới object nằm trên heap.

Quan hệ giữa heap và stack

Việc gán giá trị của biến kiểu value cho một biến khác sẽ tạo ra bản sao trong stack. Nghĩa là bạn sẽ có hai biến khác nhau nhưng có giá trị bằng nhau. Thay đổi giá trị của biến này sẽ không ảnh hưởng đến giá trị của biến kia.

Việc gán giá trị của biến kiểu reference cho một biến khác sẽ chỉ gán địa chỉ của ô nhớ. Nói cách khác, khi này hai biến sẽ trỏ vào cùng một object. Thay đổi giá trị của biến này thì biến khác đồng thời nhận sự thay đổi đó, vì thực chất chúng là cùng một object, chỉ là mang hai tên khác nhau.

Sự khác nhau này rất quan trọng khi bạn bắt đầu sử dụng biến hoặc truyền biến làm tham số cho phương thức. Không hiểu sự khác biệt này bạn có thể mắc lỗi nghiêm trọng khi gán và thay đổi giá trị của biến.

`string` mặc dù là một kiểu reference nhưng hoạt động hơi khác biệt so với các kiểu reference khác. Nó thuộc một nhóm kiểu có tên gọi là immutable. Mọi thao tác chỉnh sửa trên string đều tạo ra một object khác, thay vì thay đổi giá trị của chính object đó. Bạn sẽ học kỹ hơn về string ở một bài khác.

Giá trị null, kiểu nullable

Các biến kiểu reference có một giá trị đặc biệt, biểu diễn bằng từ khóa `null`. Giá trị này báo hiệu rằng biến reference chưa trỏ vào một object nào. Mọi thao tác xử lý trên biến reference (trừ phép gán và khởi tạo) đều báo lỗi ở giai đoạn runtime.

Tất cả biến thuộc kiểu tham chiếu khi khai báo đều nhận *giá trị mặc định* là `null`. Giá trị `null` của một biến kiểu tham chiếu thể hiện rằng biến đó chưa chứa địa chỉ của vùng nhớ nơi lưu giá trị. Chỉ khi biến đó được khởi tạo nó mới tham chiếu sang vùng lưu giá trị.

Mọi thao tác trên các biến tham chiếu (truy xuất thành viên) có giá trị `null` đều gây lỗi `NullReferenceException` *("Object reference not set to an instance of an object.")*.

C# quy định tất cả các biến kiểu tham chiếu bắt buộc phải được khởi tạo trước khi sử dụng (truy xuất các thành viên của nó).

Vì một biến kiểu giá trị không lưu địa chỉ của vùng nhớ heap mà lưu trực tiếp giá trị trong stack, biến loại này không thể nhận giá trị null.

Tuy nhiên, từ C# 2.0, bạn có thể để cho kiểu value nhận giá trị null bằng cách thêm modifier ? vào sau tên kiểu value:

```
int? count = null;
```

Modifier ? biến kiểu value thông thường thành một loại kiểu dữ liệu đặc biệt có tên là **nullable type**.

Sỡ dĩ C# phải đưa nullable type vào là vì khi làm việc với một số hệ thống ngoài, ví dụ, khi lập trình cơ sở dữ liệu với ado.net, sql server cho phép một trường nhận giá trị null. Khi đó C# bắt buộc phải có cơ chế tương ứng để thể hiện rằng biến tương ứng với giá trị null của sql phải "không có giá trị". Đối với kiểu reference, điều này hoàn toàn tương đương với việc biến đó có giá trị null. Nhưng đối với kiểu số chẳng hạn, bình thường bạn không thể biểu diễn ý tưởng "số không có giá trị". Kiểu nullable được đưa vào là để giải quyết những vấn đề đó.

Bạn sẽ học chi tiết hơn về kiểu nullable trong bài học về các kiểu dữ liệu đặc biệt của C#.

Kiểu class – struct – enum – interface – delegate

Ở một khía cạnh khác, trong .NET (và C#), kiểu dữ liệu cũng được phân loại theo vào năm nhóm: class, struct, enum, interface, delegate. Đây là cách phân loại theo chức năng và mục đích sử dụng.

Có thể bạn chưa biết, Console mà bạn đã gặp qua chương trình Hello world thuộc nhóm class, trong khi int thuộc nhóm struct. Enum, Interface và Delegate bạn sẽ gặp sau.

Tất cả các kiểu dữ liệu cơ bản ở trên đã gặp (trừ string và object) đều thuộc nhóm struct. Object và string thuộc nhóm class.

Nếu ghép nối với cách phân loại thứ nhất, tất cả các class, interface và delegate thuộc nhóm reference, struct va enum thuộc nhóm value.