Các toán tử (operator) cơ bản trong C#

Toán tử (operator, còn gọi là phép toán) là những thành phần cơ bản trong C# cũng như bất kỳ ngôn ngữ lập trình nào. Toán tử và toán hạng (operand) tạo ra các biểu thức (expression). Mỗi kiểu dữ liệu của C# có những toán tử riêng. Một phần các toán tử cơ bản của C# tương tự như trong C/C++. Tuy nhiên, C# cũng có rất nhiều toán tử đặc biệt của riêng mình. Qua mỗi phiên bản C# lại đưa thêm vào những toán tử mới.

Bài học này sẽ giới thiệu những toán tử cơ bản của C#. Những toán tử đặc biệt sẽ được xem xét chi tiết ở bài học phù hợp.

NỘI DUNG CỦA BÀI [Ẩn]

Toán tử trong C#

C# có khá nhiều toán tử (phép toán). Qua mỗi phiên bản C# lại đưa thêm vào những phép toán mới sử dụng với kiểu dữ liệu mới.

Phần lớn các toán tử cơ bản đều tương tự như các ngôn ngữ kiểu C, bao gồm các phép toán số học, phép toán logic, phép toán tăng giảm, phép toán nhị phân, phép toán index (truy xuất mảng), hay phép toán điều kiện.

Tuy nhiên, C# có nhiều phép toán hoàn toàn khác với C/C++, ví dụ phép toán kiểm tra giá trị null (null coalescing), kiểm tra kiểu, định danh, các phép toán cho delegate, v.v..

Thậm chí cho cùng một công việc nhưng phép toán của C# không giống như C/C++. Ví dụ phép toán truy xuất thành viên (object và struct).

Dưới đây là danh sách tất cả các phép toán hiện có trong C#.

NHÓM	TOÁN TỬ
Phép toán số học	+- * / %
Phép toán logic và nhị phân	& | ^ ~ && || !
Phép toán ghép xâu	+
Phép toán tăng giảm	++- --
Phép toán dịch bit	<< >>
Phép toán so sánh	== != < > <= >=
Phép gán	= += -= *= /= %= &= |= ^= <<= >>=
Phép toán truy xuất thành viên (object và struct)	.
Phép toán indexer (cho mảng)	[]
Ép kiểu (type casting)	()
Phép toán điều kiện	?:
Phép toán cho delegate (thêm/bớt)	+ -
Khởi tạo object	new
Lấy thông tin về kiểu dữ liệu	sizeof is typeof as
Kiểm soát lỗi tràn bộ đệm	checked unchecked
Phép toán liên kết null	??
Phép toán kiểm tra điều kiện null	?. ?[]
Lấy tên của phần tử	nameof()

Chắc rằng bạn sẽ thấy danh sách này vừa quen vừa lạ. Chúng ta sẽ không đi sâu vào tất cả các phép toán trên trong bài học này mà sẽ chỉ xem xét phép toán sử dụng được với các kiểu dữ liệu cơ sở của C#.

Các phép toán số học trên các kiểu số

Các phép toán số học trong C# hoàn toàn tương tự như trong C/C++. Dưới đây là danh sách các phép toán số học của C#:

Phép toán	Ví dụ
Số dương	`+x`

Số âm	`-x`
Tăng sau (post increment)	`x++`
Giảm sau (post decrement)	`x--`
Tăng trước (pre-increment)	`++x`
Giảm trước (pre-decrement)	`--x`
Nhân	`x * y`
Chia	`x / y`
Chia lấy dư	`x % y`
Cộng	`x + y`
Trừ	`x - y`

Nếu bạn xuất phát từ một ngôn ngữ không thuộc họ C, các phép toán khó hiểu nhất có lẽ là các phép toán tăng giảm (increment, decrement). Các phép toán số học còn lại đều tương tự trong các ngôn ngữ lập trình. Chúng ta sẽ nói kỹ hơn về các phép toán tăng giảm một chút.

Các phép toán tăng giảm khi sử dụng trong biểu thức sẽ cộng hoặc trừ đi một (1) đơn vị của biến khi tính toán biểu thức. Tuy nhiên, có sự khác biệt giữa tăng/giảm trước và tăng/giảm sau.

Đối với các phép tăng/giảm sau, giá trị của giá trị của toán hạng sẽ chỉ bị thay đổi **sau** khi thực hiện biểu thức.

Ví dụ, nếu x = 5 và thực hiện y = x++, y sẽ nhận giá trị 5, còn x sẽ bằng 6. Lý do là giá trị của x sẽ thay đổi (cộng thêm 1) SAU khi thực hiện biểu thức (ở đây là biểu thức gán). Do đó y sẽ nhận giá trị của x (=5) trước, sau đó x tự cộng thêm 1 để bằng 6.

Dưới đây là kết quả thực hiện trên C# Interactive.

```
> var x = 5;
> var y = x++;
> x
6
> y
5
>
```

Đối với phép tăng giảm trước, giá trị của của toán hạng sẽ tăng 1 đơn vị **trước** khi thực hiện biểu thức. Hãy xem ví dụ sau:

```
> var x = 5;
> var y = ++x;
> x
6
> y
6
>
```

tức là x sẽ tăng 1 đơn vị trước, sau đó mới thực hiện biểu thức gán. Do đó cả x và y đều có giá trị 6.

Phép toán trên bit

Dưới đây là các phép toán thực hiện trên bit – biểu diễn nhị phân của số nguyên.

Tên	Ví dụ
Phép bù (Bitwise negation)	~x
Phép và (Bitwise AND)	x & y
Phép hoặc (Bitwise OR)	x | y
Phép Bitwise XOR	x ^ y
Dịch trái (Shift left)	x << y
Dịch phải (Shift right)	x >> y

Các toán tử nhị phân này giống như trong C.

Nhiều bạn thường nhầm lẫn các phép toán này với các phép toán logic (xem phần dưới).

Lưu ý rằng, các phép toán này chỉ áp dụng với số nguyên, chính xác hơn là ở dạng biểu diễn nhị phân của số nguyên. Vì vậy nó mới có tên là bitwise operators. Dưới đây là một số ví dụ đơn giản để minh họa:

```
AND nhị phân:
    0101 (số thập phân 5)
AND 0011 (số thập phân 3)
  = 0001 (số thập phân 1)

Bù/Đảo bit:
~ 0111 (số thập phân 7)
= 1000 (số thập phân 8)

OR nhị phân:
   0101 (số thập phân 5)
OR 0011 (số thập phân 3)
 = 0111 (số thập phân 7)

XOR nhị phân:
    0101 (số thập phân 5)
XOR 0011 (số thập phân 3)
  = 0110 (số thập phân 6)
```

Một số ví dụ về phép toán dịch bit (trên số nguyên 8 bit):

```
  00010111 << 1 (số thập phân +23) Dịch chuyển trái 1 bit
= 00101110 (số thập phân +46), tương đương với nhân 2

  00010111 << 2 (số thập phân +23) Dịch sang trái 2 lần.
= 01011100 (số thập phân +92), tương đương nhân với 4 (2^2)
```

Bạn cũng có thể thử nghiệm các phép toán trên với C# Interactive:

```
> byte i = 23;
> Convert.ToString(i, 2)
"10111"
> var j = i << 1;
> j
46
> Convert.ToString(j, 2)
"101110"
>
```

Phương thức Convert.ToString(value, base) sẽ chuyển giá trị value sang hệ cơ số base.

Nếu chưa biết hết các phép toán trên là gì và có ứng dụng gì, mời bạn đọc bài viết trên wikipedia.

Các phép toán logic trên kiểu bool

Tên	Ví dụ
Logical negation (NOT)	!x
Conditional AND	x && y
Conditional OR	x || y

Các toán tử logic này hoàn toàn giống như trong C.

Lưu ý rằng, các phép toán này hoạt động trên các giá trị logic (kiểu bool) và ghép chúng lại thành các biểu thức logic phức tạp hơn. Đừng nhầm lẫn với các toán tử nhị phân ở phần trên (chỉ hoạt động trên số nguyên).

a	**b**	**!a**	**a && b**	**a || b**
true	true	false	true	true
true	false	false	false	true
false	true	true	false	true
false	false	true	false	false

Bạn cũng có thể thử nghiệm các phép toán này trên C# Interactive như sau:

```
> true && true
true
> true || false
true
> var a = false; var b = false;
> a && b
false
> a || b
false
>
```

Các phép toán logic này rất thường được sử dụng để kết hợp các biểu thức so sánh.

Các phép toán so sánh

Các phép toán so sánh (relational operator), còn gọi là các phép toán quan hệ, thực hiện được trên nhiều kiểu dữ liệu nhưng kết quả trả về luôn là kiểu bool. Các phép so sánh có thể thực hiện trên các kiểu số, kiểu ký tự và chuỗi.

Tên	Ví dụ
Nhỏ hơn	x < y
Lớn hơn	x > y
Nhỏ hơn hoặc bằng	x <= y
Lớn hơn hoặc bằng	x >= y
So sánh bằng	x == y
Không bằng	x != y

Việc so sánh này phụ thuộc một phần vào kiểu dữ liệu của toán hạng.

Nếu các toán hạng đều là số, ý nghĩa của các phép toán so sánh không khác gì trong toán học.

Nếu là ký tự, mã của ký tự đó (vốn là kiểu số nguyên) sẽ được so sánh với nhau.

Nếu là chuỗi, từng ký tự trong chuỗi sẽ được so sánh với nhau. Chúng ta sẽ quay lại so sánh chuỗi ở một bài khác.

Dưới đây là một số ví dụ về sử dụng các phép toán so sánh trong C# Interactive.

```
> int a = 10, b = 20;
> a < b
true
> a > b
false
> a != b
true
> a == b
false
>
```

Trong số các phép toán này, nếu bạn xuất phát từ ngôn ngữ khác C thì nên lưu ý phép so sánh bằng (==) và khác (!=). Ví dụ Pascal sử dụng dấu bằng = cho phép so sánh, và := cho phép gán. C# sử dụng = cho phép gán, == cho phép so sánh bằng.

Phép toán điều kiện

Phép toán điều kiện (conditional operator, ternary operator) là một đặc sản của các ngôn ngữ tương tự C. Nếu bạn biết C, bạn chắc chắn đã biết phép toán này. Nếu chưa biết, hãy cùng xem ví dụ đơn giản sau:

```
> int x = 10, y = 20;
> int z = (x > y) ? x : y;
> z
20
>
```

Đây là các lệnh để thực hiện một nhiệm vụ đơn giản: chọn giá trị lớn hơn trong hai số x y và gán cho một biến z khác.

Có thể diễn giải biểu thức int z = (x > y) ? x : y; như sau: so sánh x và y; nếu x lớn hơn y thì biểu thức có giá trị bằng x; ngược lại, biểu thức sẽ có giá trị bằng y.

Trong cú pháp của phép toán này, dấu chấm hỏi là bắt buộc và đứng sau biểu thức logic; dấu hai chấm là bắt buộc để phân tách giá trị trả về trong hai trường hợp của biểu thức logic. Nếu biểu thức logic nhận giá trị true, biểu thức sẽ nhận giá trị trước dấu hai chấm; nếu biểu thức logic có giá trị false, biểu thức sẽ nhận giá trị đứng sau dấu hai chấm. Vì đây là một biểu thức, kết quả của nó có thể gán cho một biến để sau tái sử dụng.

> Phép toán điều kiện hoạt động gần giống như cấu trúc điều kiện if-else.

Hãy cùng xem vài ví dụ khác:

```
> int a = 10, b = 20;
. string message = (a > b) ? "a lớn hơn b" : "a nhỏ hơn b";
> message
"a nhỏ hơn b"
>

> string str = "";
> (str == "") ? "Xâu rỗng" : "Xâu không rỗng"
"Xâu rỗng"
>
```

Các phép gán phức hợp

Các phép gán phức hợp (compound assignment) là nhóm phép toán đặc sản của các ngôn ngữ trong họ C. Trong đó, phép toán này thực hiện một thao tác (như cộng, trừ, nhân, chia, v.v.) và gán ngược giá trị đã biến đổi về cho biến.

Dưới đây là một số phép toán gán phức hợp.

Phép toán	Ví dụ
+=	x += 1 // tương đương x = x + 1
-=	x -= 1 // tương đương x = x – 1
*=	x *= 2 // tương đương x = x * 2
/=	x /= 2 // tương đương x = x / 2

Bạn có thể thử các phép toán này trong C# Interactive:

```
> var x = 2;
> x += 1
3
> x *= 2
6
> x /= 3
2
> x -= 2
0
>
```

Không có khó khăn gì để hiểu các phép toán này.

Các phép toán với kiểu dữ liệu: type casting, `is` và `as`, `typeof`

Type casting

Type casting (tạm dịch là ép kiểu) là việc chuyển đổi giá trị của một biến sang một kiểu khác nhưng không làm thay đổi bản chất giá trị của nó. Ví dụ, C# tự động chuyển đổi giá trị giữa các kiểu số, như từ số nguyên sang số thực và ngược lại. Loại ép kiểu này được gọi là **implicit casting**.

Tuy nhiên, trong nhiều trường hợp C# không thể tự thực hiện được mà bạn phải tự mình chỉ định kiểu đích. Loại ép kiểu này được gọi là **explicit casting**.

> Lưu ý, nếu bạn chuyển đổi từ chuỗi "1234" thành số 1234 hay ngược lại, dữ liệu đã bị thay đổi về bản chất. Đây được gọi là **type conversion.**

Để thực hiện type casting, bạn cần dùng phép toán casting theo cách sau:

```
(<kiểu-đích) <giá-trị>
```

Biểu thức này sẽ thực hiện chuyển đổi <giá-trị> sang <kiểu-đích>. Nếu quá trình ép kiểu không thành công, biểu thức sẽ phát ra ngoại lệ (exception).

Ví dụ sau đây sẽ ép kiểu của biến o1 (kiểu object) sang kiểu string và gán vào biến str1:

```
1.  > object o1 = "Hello world";
2.  > string str1 = (string)o1;
3.  > str1
4.  "Hello world"
```

Type casting với phép toán **as**

Một cách khác để thực hiện ép kiểu là sử dụng phép toán `as`. Phép toán `as` thực hiện ép kiểu cho giá trị. Nếu không thành công sẽ trả về giá trị null. Phép toán này an toàn hơn so với sử dụng phép toán ép kiểu trực tiếp ở trên do nó tránh được exception khi ép kiểu không thành công.

```
1.  > object o1 = "Hello world";
2.  > string str2 = o1 as string;
3.  > str2
4.  "Hello world"
5.  > object o2 = 12345;
6.  > string str3 = o2 as string;
7.  > str3
8.  null
9.  >
```

Tuy nhiên, phép toán as lại chỉ có thể áp dụng được đối với các kiểu tham chiếu (reference type). Nó không áp dụng được với kiểu giá trị (value type). Lý do là vì trong trường hợp ép kiểu không thành công, nó trả về giá trị `null`. Đây là giá trị đặc trưng riêng của kiểu tham chiếu (và kiểu **nullable**).

Kiểm tra kiểu – phép toán **is**

Để đảm bảo không gây lỗi khi ép kiểu, bạn nên kiểm tra kiểu (type checking) trước khi thực hiện.

C# sử dụng phép toán `is` để kiểm tra kiểu của một giá trị (object). Hãy thực hiện một vài ví dụ trên C# interactive:

```
> object o1 = "Hello world";
> o1 is string
true
> o1 is int
false
```

Phép toán `is` nhận một giá trị ở bên trái và tên kiểu ở bên phải. Nó trả về giá trị true nếu giá trị thuộc về kiểu đó. Trong ví dụ trên, giá trị của o1 thuộc kiểu string nên biểu thức `o1 is string` trả về giá trị `true`, còn `o1 is int` trả về giá trị `false`.

Lấy thông tin về kiểu: phép toán **typeof**

Phép toán **typeof** trả về một object chứa thông tin về kiểu dữ liệu. Từ kết quả này bạn có thể lấy tất cả các thông tin cần thiết về chính kiểu dữ liệu. Ví dụ:

```
> Type stringType = typeof(string);
> stringType
[System.String]
> stringType.UnderlyingSystemType
[System.String]
> stringType.Assembly
[mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089]
> stringType.BaseType
[System.Object]
> stringType.FullName
"System.String"
```

Lưu ý rằng, typeof chỉ nhận tham số là tên kiểu dữ liệu. Nếu muốn lấy thông tin về kiểu từ biến, bạn phải dùng phương thức GetType():

```
// Lưu ý: nếu bạn muốn lấy thông tin về kiểu dữ liệu từ biến, bạn phải dùng phương thức
> stringType = "Hello world".GetType();
// hai phương pháp này cho cùng một kết quả
```

Phép toán typeof được sử dụng rất nhiều cùng với lập trình dynamic, generic và reflection.

Kết luận

Trong bài học này bạn đã tiếp xúc với các phép toán (toán tử) cơ bản của C#. Rất dễ nhận thấy rằng, các toán tử này hoàn toàn tương tự như trong các ngôn ngữ kế thừa cú pháp của ngôn ngữ C.

Nếu bạn có xuất phát điểm là một trong những ngôn ngữ này, việc nắm bắt các toán tử cơ bản của C# rất đơn giản. Ngay cả khi bạn xuất phát từ một ngôn ngữ khác, các phép toán cơ bản của C# không có gì phức tạp. Các vấn đề cần lưu ý đã được trình bày trong bài.

Cũng lưu ý rằng, nhiều toán tử khác của C# chưa được trình bày trong bài. Chúng ta sẽ làm việc với chúng ở những bài học phù hợp.

+ Nếu bạn thấy bài viết hữu ích, hãy giúp **chia sẻ** tới mọi người.
+ Nếu có thắc mắc hoặc cần trao đổi thêm, mời bạn viết trong phần **thảo luận** cuối trang.

Cảm ơn bạn!