Documento Operadores

Operadores de C#

C# proporciona muchos operadores, que son símbolos que especifican las operaciones (matemáticas, indización, llamada de función, etc.) que se realizan en una expresión. Puede sobrecargar muchos operadores para cambiar su significado al aplicarlos a un tipo definido por el usuario.

Las operaciones de tipos enteros (como ==, !=, <, >, & y |) suelen estar permitidas en los tipos de enumeración (enum).

En las secciones siguientes se enumeran los operadores de C# desde la precedencia más alta a la más baja. Los operadores de cada sección comparten el mismo nivel de precedencia.

Operadores principales

Estos son los operadores de precedencia más alta.

x.y: acceso a miembros.

x?.y: acceso a miembros condicionales nulos. Devuelve null si el operando izquierdo se evalúa como null.

x?[y]: acceso a índices condicionales nulos. Devuelve null si el operando izquierdo se evalúa como null.

f(x): invocación de función.

a[x]: indización de objeto agregado.

x++: incremento de postfijo. Devuelve el valor de x y, a continuación, actualiza la ubicación de almacenamiento con el valor de x que es uno mayor (normalmente agrega el entero 1).

x--: decremento de postfijo. Devuelve el valor de x; a continuación, actualiza la ubicación de almacenamiento con el valor de x que es uno menos (normalmente resta el entero 1).

new: creación de instancias de tipo.

typeof: devuelve el objeto Type que representa el operando.

checked: habilita la comprobación de desbordamiento para operaciones con enteros.

unchecked: deshabilita la comprobación de desbordamiento para operaciones con enteros. Este es el comportamiento predeterminado del compilador.

default(T) genera el valor predeterminado del tipo T.

delegate: declara y devuelve una instancia de delegado.

sizeof: devuelve el tamaño en bytes del operando de tipo.

->: desreferenciación del puntero combinada con acceso a miembros.

Operadores unarios

Estos operadores tienen mayor precedencia que los de la sección siguiente y menor que el de la anterior.

+x: devuelve el valor de x.

-x: negación numérica.

!x: negación lógica.

~x: complemento bit a bit.

++x: incremento de prefijo. Devuelve el valor de x después de actualizar la ubicación de almacenamiento con el valor de x que es uno mayor (normalmente agrega el entero 1).

--x: decremento de prefijo. Devuelve el valor de x después de actualizar la ubicación de almacenamiento con el valor de x que es uno menos (normalmente resta el entero 1).

(T)x: conversión de tipos.

await: espera una Task.

&x: dirección de.

*x: desreferenciación.

Operadores de multiplicación

Estos operadores tienen mayor precedencia que los de la sección siguiente y menor que el de la anterior.

x * y: multiplicación.

x / y: división. Si los operandos son enteros, el resultado es un entero que se trunca hacia cero (por ejemplo, -7 / 2 is -3).

x % y: resto. Si los operandos son enteros, devuelve el resto de dividir x entre y. Si q = x / y y r = x % y, entonces x = q * y + r.

Operadores aditivos

Estos operadores tienen mayor precedencia que los de la sección siguiente y menor que el de la anterior.

x + y: suma.

x – y: resta.

Operadores de desplazamiento

Estos operadores tienen mayor precedencia que los de la sección siguiente y menor que el de la anterior.

x << y: desplaza los bits a la izquierda y rellena con cero a la derecha.

x >> y: desplaza los bits a la derecha. Si el operando izquierdo es int o long, los bits de la izquierda se rellenan con el bit de signo. Si el operando izquierdo es uint o ulong, los bits de la izquierda se rellenan con cero.

Operadores de comprobación de tipos y relacionales

Estos operadores tienen mayor precedencia que los de la sección siguiente y menor que el de la anterior.

x < y: menor que (true si x es menor que y).

x > y: mayor que (true si x es mayor que y).

x <= y: menor o igual que.

x >= y: mayor o igual que.

is: compatibilidad de tipos. Devuelve true si el operando izquierdo evaluado se puede convertir al tipo especificado en el operando derecho (un tipo estático).

as: conversión de tipos. Devuelve el operando izquierdo convertido al tipo especificado por el operando derecho (un tipo estático), pero as devuelve null donde (T)x produciría una excepción.

Operadores de igualdad

Estos operadores tienen mayor precedencia que los de la sección siguiente y menor que el de la anterior.

x == y: igualdad. De forma predeterminada, para los tipos de referencia distintos de string, devuelve igualdad de referencia (prueba de identidad). Sin embargo, los tipos pueden sobrecargar ==, por lo que si su intención es probar la identidad, es mejor usar el método ReferenceEquals en object.

x != y: distinto de. Vea el comentario de ==. Si un tipo sobrecarga ==, debe sobrecargar !=.

AND lógico (operador)

Este operador tiene mayor precedencia que el de la sección siguiente y menor que el de la anterior.

x & y: AND lógico o bit a bit. Por lo general puede usarlo con tipos enteros y tipos enum.

Operador lógico XOR

Este operador tiene mayor precedencia que el de la sección siguiente y menor que el de la anterior.

x ^ y: XOR lógico o bit a bit. Por lo general puede usarlo con tipos enteros y tipos enum.

Operador lógico OR (||)

Este operador tiene mayor precedencia que el de la sección siguiente y menor que el de la anterior.

x | y: OR lógico o bit a bit. Por lo general puede usarlo con tipos enteros y tipos enum.

Operador condicional AND

Este operador tiene mayor precedencia que el de la sección siguiente y menor que el de la anterior.

x && y: AND lógico. Si el primer operando se evalúa como false, C# no evalúa el segundo operando.

Operador condicional OR

Este operador tiene mayor precedencia que el de la sección siguiente y menor que el de la anterior.

x || y: OR lógico. Si el primer operando se evalúa como true, C# no evalúa el segundo operando.

Operador de uso combinado de null

Este operador tiene mayor precedencia que el de la sección siguiente y menor que el de la anterior.

x ?? y: devuelve x si no es null; de lo contrario, devuelve y.

Operador condicional

Este operador tiene mayor precedencia que el de la sección siguiente y menor que el de la anterior.

t ? x : y: si la prueba t se evalúa como true, evalúa y devuelve x; en caso contrario, evalúa y devuelve y.

Operadores de asignación y Lambda

Estos operadores tienen mayor precedencia que los de la sección siguiente y menor que el de la anterior.

x = y: asignación.

x += y: incremento. Agregue el valor de y al valor de x, almacene el resultado en x y devuelva el nuevo valor. Si x designa un event, y debe ser una función adecuada que C# agregue como un controlador de eventos.

x -= y: decremento. Reste el valor de y del valor de x, almacene el resultado en x y devuelva el nuevo valor.Si x designa un event, y debe ser una función adecuada que C# quite como un controlador de eventos.

x *= y: asignación de multiplicación. Multiplique el valor de y por el valor de x, almacene el resultado en xy devuelva el nuevo valor.

x /= y: asignación de división. Divida el valor de x por el valor de y, almacene el resultado en x y devuelva el nuevo valor.

x %= y: asignación del resto. Divida el valor de x por el valor de y, almacene el resto en x y devuelva el nuevo valor.

x &= y: asignación de AND. AND el valor de y con el valor de x, almacene el resultado en x y devuelva el nuevo valor.

x |= y: asignación de OR. OR el valor de y con el valor de x, almacene el resultado en x y devuelva el nuevo valor.

x ^= y: asignación de XOR. XOR el valor de y con el valor de x, almacene el resultado en x y devuelva el nuevo valor.

x <<= y: asignación de desplazamiento a la izquierda. Desplace el valor de x a la izquierda y lugares, almacene el resultado en x y devuelva el nuevo valor.

x >>= y: asignación de desplazamiento a la derecha. Desplace el valor de x a la derecha y posiciones, almacene el resultado en x y devuelva el nuevo valor.

=>: declaración lambda.