Cómo... en .NET

Operar a nivel de bits (operaciones lógicas a nivel de bit)

Publicado el 12/May/2008
Actualizado el 12/May/2008
Autor: Guillermo 'guille' Som

En este artículo te explico cómo operar con valores a nivel de bits, para hacer operaciones con los operadores AND, OR, NOT y XOR. Y para ver cómo funcionan estos valores a niveles de bits, te pongo un ejemplo para acceder a los atributos de los ficheros. Y como de costumbre, con código para Visual Basic y C#.

Introducción con batallita del agüelo incluida:

Este tipo de operaciones "lógicas" no son muy habituales, pero seguro que te las encontrarás más de una vez.

Si te contara una batallita te diría que..., ¡vale!, pues te la cuento, total, esto que te voy a explicar se explica en cuatro líneas.

En "mis tiempos", debido a que teníamos que ahorrar memoria y espacio en disco, (por si no lo sabes, en los primeros tiempos de los PC-Compatibles los discos eran de 360 KB y había que aprovechar bien el contenido), había veces que en lugar de guardar valores normales, lo que hacíamos era almacenar valores que tenían ciertos bits conectados o no, y dependiendo

valores que estaban guardados en el disco, y de esta forma, sabiendo si el bit estaba a uno o a cero sabía si en ese "registro" había información o no. Ahora esto es casi inútil hacerlo, ya que no hay problemas, ni de almacenamiento ni de velocidad, pero... en fin...

Pues eso, un byte son 8 bits y cuando trabajamos a nivel de bits lo que hacemos es trabajar a nivel binario, y a ese nivel solo existen dos valores posibles: cero y uno.

Cuando quieres trabajar con esos valores, puedes hacerlo usando lo que se conoce como **aritmética binaria** (o a nivel de bits).

Por ejemplo, si sumas uno al valor 3 tienes un 4. Esto es de primaria, pero si te lo digo en binario, ya no es tan "lógico" ni evidente:

0011 + 0001 = 0100. ¿Está fácil? Pues no tanto... salvo que sepas sumar solo con dos dedos ;-)))

De todas formas te explico lo que ha ocurrido. Empezando a operar con los valores desde la derecha (según se mira la pantalla):

```
0011
0001
----
0100
```

1 + 1 = 0 y me llevo 1 (ya que el 2 decimal sería el 10 binario).
1 + 0 + 1 = 0 y me llevo 1.
0 + 0 + 1 = 1.
0 + 0 = 0.
Y uniéndolo todo, tenemos 0100.

Pero esto no es el tema de lo que te quería contar, pero para que vayas viendo cuánto nos podemos complicar si nos salimos del sistema decimal (o base 10).

El tema que te quería contar era operar con esos bits haciendo operaciones "lógicas", es decir, usando los operadores OR, AND, NOT y XOR. Los pongo así en mayúsculas para que te sirva tanto para Visual Basic como para C#, aunque en los lenguajes que tienen su raíz en C, son más complicados para estas cosas (tan evidentes), por ejemplo, en esos lenguajes de la familia C, para la operación AND a nivel de bit se utiliza el símbolo &. Para realizar una operación OR se usa |. Para realizar una operación NOT se usa !. Y para usar una operación XOR se usa ^. XOR es OR exclusivo.

Si queremos operar a nivel de bits, podemos usar esas cuatro operaciones que en la siguiente tabla tienes el resultado obtenido al operar con cada una de las posibilidades que nos da el sistema binario.

Para cada tipo de operación en la parte de la izquierda tienes la operación usando los valores **Boolean**, en la parte de la derecha tienes los valores de

Nota para los de C#:
El operador And es &
Or es |
Not es !
Xor es ^

AND **Devuelve un valor verdadero, si las dos expresiones devuelven un valor verdadero.**

True And True = True	(1 And 1 = 1)
True And False = False	(1 And 0 = 0)
False And True = False	(0 And 1 = 0)
False And False = False	(0 And 0 = 0)

NOT **Niega (o invierte) el valor devuelto por la expresión.**

Not True = False	(Not 1 = 0)
Not False = True	(Not 0 = 1)

OR **Devuelve un valor verdadero si cualquiera de las dos expresiones es verdadera.**

True Or True = True	(1 Or 1 = 1)
True Or False = True	(1 Or 0 = 1)
False Or True = True	(0 Or 1 = 1)
False Or False = False	(0 Or 0 = 0)

XOR **Devuelve un valor verdadero solamente si una de las expresiones es verdadera.**

True Xor True = False	(1 Xor 1 = 0)
True Xor False = True	(1 Xor 0 = 1)
False Xor True = True	(0 Xor 1 = 1)
False Xor False = False	(0 Xor 0 = 0)

Tabla 1. Operaciones binarias a nivel lógico y a nivel de bits.

Seguramente dirás: Vale, mu bonito... ¿y?
Pues... a ver... si por un lado tienes el valor 3 (que en binario es 0011) y por otro tienes el valor 10 (que en binario es 1010) y le aplicas estos operadores tendrás:

```
    0011 (3)
Or  1010 (10)
--------
    1011 (11)
```

```
    0011 (3)
And 1010 (10)
--------
    0010 (2)
```

```
    0011 (3)
Xor 1010 (10)
--------
    1001 (9)
```

Debes tener en cuenta que para los valores binarios, la posición que ocupa es la que le da el valor, por ejemplo para 8 bits (1 byte), tenemos estas posiciones y los valores correspondientes a cada bit:

```
Bit     8  7  6  5 4 3 2 1
Valor 128 64 32 16 8 4 2 1
```

En realidad cada valor se corresponde con 2 elevado a la posición, por ejemplo: 2 elevado a 8 = 128.

Por tanto, si tenemos el valor del 9 (00001001), tenemos un 1 en la posición del valor 8 y otro en la posición del valor 1:

```
Bit     8  7  6  5 4 3 2 1
Valor 128 64 32 16 8 4 2 1
        0  0  0  0 1 0 0 1
```

¿Que valor tendrá este número?

```
Bit     8 7 6 5 4 3 2 1
        1 0 1 0 1 0 1 0
```

¿Para que sirve todo esto?

Pues para operar a niveles de bits.
Ah... ¿te refieres de forma práctica?

Por ejemplo cuando trabajas con valores de una enumeración que tiene el atributo **Flags** y cuyos valores se correspondan con valores binarios, por ejemplo, que un valor sea 1, el siguiente 2, el siguiente 4 y el siguiente 8, etc. Es decir 2 elevado a la potencia de la posición que ocupa, que es en realidad el valor que tiene cada bit del número mostrado antes.

Como algo práctico, los valores de los atributos de los ficheros (o directorios) suelen tener valores de este tipo.
Por ejemplo, si es un directorio tiene activado el bit correspondiente al valor 16 (el bit de la posición 5).
Si está oculto el bit que tendrá activado es el 2.
Por tanto, si el valor del atributo que tiene un directorio es 18 será porque es un directorio (16) y está oculto (2).
El valor 18 en binario sería: 10010

Y con el AND podemos saber si un bit en cuestión está activado, ya que si hacemos esta comparación:
Valor And Posición = Posición (para C#, sería: Valor & Posición == Posición)
Y se cumple, querrá decir que Valor tiene el valor indicado por Posición.

Vale, mejor con algo práctico.

Si tenemos el siguiente código, podemos saber si ese fichero está oculto:

```
Dim atr As FileAttributes = File.GetAttributes(fic)

If (atr And FileAttributes.Hidden) = FileAttributes.Hidden Then
    ' Está oculto
Else
    ' No está oculto
End If
```

```
FileAttributes atr = File.GetAttributes(s);

if((atr & FileAttributes.Hidden) == FileAttributes.Hidden)
{
    // Está oculto
}
else
{
    // No está oculto
}
```

¿Por qué funciona esto como funciona?

Cuando haces un AND estás devolviendo todos los bits que en ambos lados estén a 1, y como trabajamos con valores que siempre tienen activado un bit (si no se trabajara así, esto no funcionaría), se comprueba si al hacer el AND obtenemos el mismo valor, es que ese "bit" está en el valor total del atributo.

Es decir, el atributo **Hidden** tiene un valor 2. El atributo **Directory** tiene un valor 16. El atributo **System** vale 4. El atributo **ReadOnly** vale 1. Y así sucesivamente.

Por tanto, si es un directorio oculto, del sistema y de solo lectura ¿qué valor tendrá?
16 + 2 + 4 + 1 = 23 que en binario es:

```
Bit     8  7  6  5 4 3 2 1
Valor 128 64 32 16 8 4 2 1
        0  0  0  1 0 1 1 1
```

Nota sobre OR y la suma:

La operación OR a nivel de bit se puede sustituir por una operación de suma, pero solo tendrá sentido "decimal" si los valores que sumamos son valores exactos (a nivel de bits), por ejemplo, sumar 8 + 2 es lo mismo que hacer 8 Or 2, que en ambos casos dará como resultado 10.

Pero si los valores a sumar no representan valores exactos a nivel de bits, no producirá el mismo resultado.
Por ejemplo 10 + 3 no es lo mismo que 10 Or 3, ya que al sumar será 13, pero al hacer Or será 11.
La explicación es porque 10 es: 1010 que al hacer OR con 0011 tenemos 1011 (el ejemplo que vimos al principio),
mientras que 13 es: 1101.

Esto lo digo para aquellos que han usado Visual Basic 6.0 o anterior (incluso con Visual Basic .NET y Option Strict Off), ya que Visual Basic permite usar la suma en lugar de OR para "ligar" valores que tienen valores binarios, pero como acabamos de ver esa suma solo se podrá hacer con valores binarios exactos.

Espero que te haya quedado claro y que de alguna forma te sea de utilidad.

Nos vemos.

Código de ejemplo (comprimido):

No hay proyecto de ejemplos, todo el código está en el artículo.

La fecha/hora en el servidor es: 31/10/2020 10:39:31

La fecha actual GMT (UTC) es: Sat, 31 Oct 2020 09:39:11 GMT