

Micro guía CC30A: operaciones de bits en Java

Profesor: Benjamín Bustos, Auxiliares: Hernán Arroyo, Carlos Cabrera.

11 de octubre de 2007

Java posee operadores que permiten manipulación directa de los bits de algunos tipos elementales: *int* y *long*.

BitWise operators

El primer grupo de operadores trabaja sobre la representación binaria de dos operandos, bit a bit; a continuación una lista de ellos y cómo se escriben:

Operador	Notación	Ejemplo
AND - <i>y</i>	&	0101 & 0011 = 0001
OR - <i>o</i>		0101 0011 = 0111
XOR - <i>o exclusivo</i>	^	0101 ^ 0011 = 0110
NOT - <i>no</i>	~	~0101 = 1010

Máscaras de bits

Suponga que deseamos sólo trabajar con un cierto grupo de bits dentro de otro(osea, descartar algunos bits dentro de un *int*, por ejemplo), para ello se utiliza una *máscara de bits*, que es un número en donde sólo están *encendidos* los bits con lo que nos interesa trabajar.

La máscara se aplica al número que queremos filtrar con el operador AND. A continuación un ejemplo con dos *ints*:

Número a "filtrar"	Máscara	Resultado
01100110111011101001110100110101	00000000000000000000000000000000110	00000000000000000000000000000000100

En java el ejemplo se escribiría de esta forma:

```
int num = 1726913845; /*que es el número a filtrar escrito en base decimal*/

int mask = 5;
int res = num & mask; /* res = 4 */
```

Bit Shift¹

Hay dos tipos de shifts: *shifts aritméticos* y *shifts lógicos*. El *shift lógico* desplaza una trama de bits hacia la derecha o hacia a la izquierda rellenando con ceros los espacios que quedan vacíos. El *shift aritmético* trata de conservar el signo de los números que están representados en *notación de complementos de dos*²: cuando se hace un shift hacia la derecha, y el bit de más a la izquierda (el bit *más significativo*) es un 1, se rellenan con 1's los lugares que quedan vacíos; el motivo de hacer esto es conservar el signo de un número al hacer el shift (si no se hiciera así, los números negativos al hacer el shift pasarían a positivos y con un valor que no corresponde al que uno esperaría); el *shift aritmético* hacia la izquierda es exactamente igual al *shift lógico* hacia la izquierda.

En Java todos los tipos elementales enteros(byte, short, int, long) se interpretan *con signo* (en contraste, otros lenguajes de más bajo nivel como C permiten declarar variables que el computador interpretará sin signo: *unsigned int*, *unsigned short*, *unsigned long*, *unsigned long long* ...).

La siguiente tabla muestra la notación en java para los shifts y algunos ejemplos de cada tipo:

¹http://en.wikipedia.org/wiki/Bit_shift#Bit_shifts

²Explicada en detalle acá: http://es.wikipedia.org/wiki/Complemento_a_dos

Operación	Notación	ejemplo, en decimal	ejemplo, bits
shift <i>aritmético</i> hacia la <i>izquierda</i> (= <i>lógico</i>) de x en n posiciones	$x \ll n$	$1 \ll 2 = 4$	$00001 \ll 2 = 00100$
shift <i>aritmético</i> hacia la <i>derecha</i> de x en n posiciones	$x \gg n$	$5 \gg 1 = 2$	$00101 \gg 1 = 00010$
shift <i>lógico</i> hacia la <i>derecha</i> de x en n posiciones	$x \ggg n$	$5 \ggg 1 = 2$	$00101 \ggg 1 = 00010$

Construcciones comunes con operadores de bits:

Un problema recurrente cuando se quiere trabajar directamente con los bits es testear si el bit n -ésimo está encendido:

```
int n=3; /* cambiaré el tercer bit de derecha a izquierda */
int num=13; /* num = ...01101 */
int mask=1; /* mask = ...00001 */
mask = mask << (n-1); /* mask = ...00100 */
if( (mask & num) != 0 ) /* ¿ quedó algun bit prendido ? */
    System.out.println("el bit está prendido");
else
    System.out.println("el bit está apagado");
```

Otra cosa que uno recurrentemente quiere hacer es cambiar directamente el valor del bit n -ésimo por 0:

```
int n=3; /* cambiaré el tercer bit de derecha a izquierda */
int num=13; /* num = ...01101 */
int mask=1; /* mask = ...00001 */
mask = mask << (n-1); /* mask = ...00100 */
mask = ~mask; /* mask = ...11011 */
num = num & mask; /* la máscara deja pasar a todos los bits menos al que queremos borrar*/
```

Y por 1:

```
int n=2; /* cambiaré el segundo bit de derecha a izquierda */
int num=13; /* num = ...01101 */
int mask=1; /* mask = ...00001 */
mask = mask << (n-1); /* mask = ...00010 */
num = num | mask; /* todo lo que es uno en la máscara ahora lo es también en num*/
```