Java: les opérateurs

Juin 2014

Qu'est-ce qu'un opérateur ?

Les opérateurs sont des symboles qui permettent de manipuler des variables, c'est-à-dire effectuer des opérations, les évaluer, ... On distingue plusieurs types d'opérateurs :

- les opérateurs de calcul
- les opérateurs d'assignation
- les opérateurs d'incrémentation
- les opérateurs de comparaison
- les opérateurs logiques
- (les opérateurs bit-à-bit)
- (les opérateurs de rotation de bit)

Les opérateurs de calcul

Les opérateurs de calcul permettent de modifier mathématiquement la valeur d'une variable

Opérateur	Dénomination	Effet	Exemple	Résultat (int x=7)
+	opérateur d'addition	Ajoute deux valeurs	x+3	10
-	opérateur de soustraction	Soustrait deux valeurs	x-3	4
*	opérateur de multiplication	Multiplie deux valeurs	x*3	21
/	opérateur de division	Calcul le quotient de la division de deux valeurs	x/3	2
%	opérateur de congruence	Calcul le reste de la division de deux valeurs	x%3	1
=	opérateur d'affectation	Affecte une valeur à une variable	x=3	Met la valeur 3 dans la variable x

Les opérateurs d'assignation

Ces opérateurs permettent de simplifier des opérations telles que *ajouter une valeur dans une variable et stocker le résultat dans la variable*.
Une telle opération s'écrirait habituellement de la façon suivante par exemple: *x=x+2* Avec les opérateurs d'assignation il est possible d'écrire cette opération sous la forme suivante :*x+=2*
Ainsi, si la valeur de x était 7 avant opération, elle sera de 9 après...

Les autres opérateurs du même type sont les suivants :

Opérateur	Effet
+=	addition deux valeurs et stocke le résultat dans la variable (à gauche)
-=	soustrait deux valeurs et stocke le résultat dans la variable
***=**	multiplie deux valeurs et stocke le résultat dans la variable
/=	divise deux valeurs et stocke le quotient dans la variable
%=	divise deux valeurs et stocke le reste dans la variable

Les opérateurs d'incrémentation

Ce type d'opérateur permet de facilement augmenter ou diminuer d'une unité une variable. Ces opérateurs sont très utiles pour des structures telles que des boucles, qui ont besoin d'un compteur (variable qui augmente de un en un).

Un opérateur de type *x++* permet de remplacer des notations lourdes telles que *x=x+1* ou bien *x+=1*

Opérateur	Dénomination	Effet	Syntaxe	Résultat (int x=7)
++	Incrémentation	Augmente d'une unité la variable	x++ ou ++x	8
--	Décrémentation	Diminue d'une unité la variable	x-- ou --x	6

Remarque : la différence entre x++ ou ++x se fait ressentir lorsque l'on combine plusieurs opérations. Avec x++ on utilise la valeur de x puis on ajoute 1, alors qu'avec ++x on ajoute d'abord 1, puis on utilise le résultat. Il en va de même pour x-- et --x. Exemples (pour x=7)

- `int y=x++;`

  y est initialisé à 7, puis x est incrémenté à 8

- `int y=++x;`

  x est incrémenté à 8, puis y est initialisé à 8

Les opérateurs de comparaison

Opérateur	Dénomination	Effet	Exemple	Résultat
== **À ne pas confondre avec le signe d'affectation =**	opérateur d'égalité	Compare deux valeurs et vérifie leur égalité	x==3	Retourne *true* si x est égal à 3, sinon *false*
<	opérateur d'infériorité stricte	Vérifie qu'une variable est strictement inférieure à une valeur	x<3	Retourne *true* si x est inférieur à 3, sinon *false*
<=	opérateur d'infériorité	Vérifie qu'une variable est inférieure ou égale à une valeur	x<=3	Retourne *true* si x est inférieur ou égal à 3, sinon *false*
>	opérateur de supériorité stricte	Vérifie qu'une variable est strictement supérieure à une valeur	x>3	Retourne *true* si x est supérieur à 3, sinon *false*
>=	opérateur de supériorité	Vérifie qu'une variable est supérieure ou égale à une valeur	x>=3	Retourne *true* si x est supérieur ou égal à 3, sinon *false*
!=	opérateur de différence	Vérifie qu'une variable est différente d'une valeur	x!=3	Retourne *true* si x est différent de 3, sinon *false*

Les opérateurs logiques (booléens)

Ce type d'opérateur permet de vérifier si plusieurs conditions sont vraies :

Opérateur	Dénomination	Effet	Syntaxe
||	OU logique	Retourne *true* si au moins une des deux conditions vaut *true* (ou *false* sinon)	condition1 || condition2
&&	ET logique	Retourne *true* si les deux conditions valent *true* (ou *false* sinon)	condition1 && condition2
!	NON logique	Retourne *true* si la variable vaut *false*, et *false* si elle vaut *true*)	!condition

Remarque : il n'est pas toujours nécessaire de faire tous les tests pour connaître le résultat d'un calcul. Par exemple le résultat de *(true || x)*, vaudra toujours *true* quelque soit la valeur x. Il est donc intéressant de savoir que Java évalue les valeurs de gauche à droite pour économiser les

calculs.

Les opérateurs bit-à-bit

Si vous ne comprenez pas ces opérateurs cela n'est pas important, vous n'en aurez probablement pas l'utilité. Pour ceux qui voudraient comprendre, rendez- vous aux chapitres suivants :

- compréhension du binaire
- représentation des données
- Instructions arithmétiques et logiques en assembleur

Ce type d'opérateur traite ses opérandes comme des données binaires, plutôt que des données décimales, hexadécimales ou octales. Ces opérateurs traitent ces données selon leur représentation binaire mais retournent des valeurs numériques standards dans leur format d'origine.

Les opérateurs suivants effectuent des opérations bit-à-bit, c'est-à-dire avec des bits de même poids.

Opérateur	Dénomination	Effet	Syntaxe	Résultat
&	ET bit-à-bit	Retourne 1 si les deux bits de même poids sont à 1	9 & 12 (1001 & 1100)	8 (1000)
|	OU inclusif	Retourne 1 si l'un ou l'autre des deux bits de même poids est à 1 (ou les deux)	9 | 12 (1001 | 1100)	13 (1101)
^	OU exclusif	Retourne 1 si l'un des deux bits de même poids est à 1 (mais pas les deux)	9 ^ 12 (1001 ^ 1100)	5 (0101)

Les opérateurs de rotation de bit

Si vous ne comprenez pas ces opérateurs cela n'est pas important, vous n'en aurez probablement pas l'utilité. Pour ceux qui voudraient comprendre, rendez- vous aux chapitres suivants :

- compréhension du binaire
- représentation des données
- Instructions arithmétiques et logiques en assembleur

Ce type d'opérateur traite ses opérandes comme des données binaires, plutôt que des données décimales, hexadécimales ou octales. Ces opérateurs traitent ces données selon leur représentation binaire mais retournent des valeurs numériques standards dans leur format d'origine.

Les opérateurs suivants effectuent des rotation sur les bits, c'est-à-dire qu'il décale chacun des bits d'un nombre de bits vers la gauche ou vers la droite. La première opérande désigne la donnée sur laquelle on va faire le décalage, la seconde désigne le nombre de bits duquel elle va être décalée.

Opérateur	Dénomination	Effet	Syntaxe	Résultat
<<	Rotation à gauche	Décale les bits vers la gauche (multiplie par 2 à chaque décalage). Les bits qui sortent à gauche sont perdus, tandis que des zéros sont insérés à droite	6 << 1 (110 << 1)	12 (1100)
>>	Rotation à droite avec conservation du signe	Décale les bits vers la droite (divise par 2 à chaque décalage). Les bits qui sortent à droite sont perdus, tandis que le bit non-nul de poids plus fort est recopié à gauche	6 >> 1 (0110 >> 1)	3 (0011)
>>>	Rotation à droite avec remplissage de zéros	Décale les bits vers la droite (divise par 2 à chaque décalage). Les zéros qui sortent à droite sont perdus, tandis que des zéros sont insérés à gauche	3 >>> 1 (0011 >>> 1)	1 (0001)

Les priorités

Lorsque l'on associe plusieurs opérateurs, il faut que le compilateur sache dans quel ordre les traiter, voici donc dans **l'ordre décroissant** les priorités de tous les opérateurs :

Priorité des opérateurs								
++++++++++++++++	()	[]	.					
+++++++++++++++	--	++	!	~	- (un opérande)	+ (un opérande)	(casting)	new
+++++++++++++	*	/	%					
++++++++++++	+	-						
++++++++++++	<<	>>	>>>					
++++++++++	<	<=	>=	>	instanceof			
++++++++++	==	!=						
+++++++++	&							
++++++++	^							
+++++++	|							
++++++	&&							
+++++	||							
++++	?	:						

+++	=	+=	-=	*=	/=	%=	&=	
++	<<=	>>=	>>>=	^=	|=			
+	,							