

JAVA ET LES REFERENCES, DE GESTION DE MEMOIRE

Par Richard, Directeur Technique Adneom, Consultant JEE.

Introduction

En Java, les variables sont soit de type valeur, soit de type référence.

Les types valeurs ont une taille connue à l'avance et peuvent donc être alloués sur la pile d'exécution (stack) sans risquer un `StackOverflowError`, ce qui n'est pas possible avec les types référence (classe et tableau) car la taille n'est pas maîtrisée. Ces derniers sont donc alloués dans une zone qu'on appelle le tas (heap) et la pile d'exécution ne stocke qu'une référence (une adresse) vers le tas. Allons un peu plus en détail et voyons comment se comportent ces références par rapport à la gestion de la mémoire.

Le ramasse miette

Quand Java est sorti, deux grandes nouveautés l'ont démarqué des langages existants : sa portabilité et sa gestion de la mémoire. Cette dernière est assurée par un composant bien connu des développeurs : le ramasse miette (Garbage Collector). Son fonctionnement est assez simple. En tâche de fond, celui-ci scanne l'ensemble des éléments du tas et détermine pour chacun s'il y a encore une référence active. Dans le cas où il n'y en a plus, cet élément est collecté. Ce qui semble une idée géniale (quoique pas inventé par Java) peut cependant poser quelques problèmes tels qu'une fuite mémoire ou une incapacité à gérer la mémoire de façon fine (entre autre le ramasse miette n'est pas déterministe, ce qui empêche Java d'être utilisé comme langage en temps réel).

Les différents types de références

Le mode de fonctionnement du ramasse miette présenté ci-dessus n'est en fait pas si simple car il n'y a pas un type de référence mais quatre. Il y a celle qu'on connaît tous quand par exemple on écrit `String text = new String("Hello World");`. Cette expression crée une instance de `String` qui est stockée dans le tas et à laquelle la variable `text` fait référence suite à l'affectation avec l'opérateur `=`. Cette référence est en fait une strong reference. Dans le package `java.lang.ref`, nous trouvons en plus les `SoftReference`, les `WeakReference` et les `PhantomReference`, qui autorisent une certaine interaction avec le ramasse miette.

Pour en revenir à ce dernier, le choix de collecter un objet n'est donc pas que fonction du nombre de références qui pointent dessus mais aussi du type de celles-ci. Dans le cas d'une strong reference ce qui a été dit plus haut est toujours valable, à savoir :

- S'il reste ne serait ce qu'une strong reference sur un objet, celui-ci n'est pas collecté.

Voyons donc pour les autres cas :

- S'il ne reste que des soft references sur un objet, celui-ci n'est collecté que si la JVM a besoin de mémoire (ie avant de lancer une `OutOfMemoryError`).

- S'il ne reste que des weak references sur un objet, celui-ci est collecté.

Le cas des phantom references est particulier car il concerne des objets qui sont déjà collectés. Ce genre de référence est utilisé afin de connaître l'état de la mémoire suite aux actions du ramasse miette sans interférer avec celui-ci (il ne faudrait pas recréer des références sur des objets collectés).

Problème de référence circulaire

Prenons le cas simple d'une relation père fils. Le père connaît son fils et le fils connaît son père. Coder de façon simple, nous aurions donc un objet A qui a une strong reference sur B et B qui a une strong reference sur A. Dans ces conditions comment A et B peuvent-ils être collectés ? En effet même si plus aucun autre objet n'a de strong reference sur A et B, ces deux objets se font mutuellement référence. La solution serait par exemple que A ait une strong reference sur B mais que B n'ait qu'une weak reference sur A. Ainsi quand il n'y a plus de strong reference sur A, ce dernier peut être collecté car B n'y fait référence que de façon faible.

Implémentation d'un cache

Dans les applications effectuant des requêtes qui récupèrent des objets, il y en a souvent qui sont régulièrement ramenés, afin d'optimiser ces requêtes on utilise alors un cache. Le problème qui se poserait avec un cache implémenté à base de strong reference serait : « Comment doit-on vider ce dernier afin d'éviter une OutOfMemoryError ? ». La réponse est assez simple, utiliser des soft reference. Ainsi le cache contient un maximum d'objets (conservant ainsi son efficacité), les objets ne seraient collectés que quand la mémoire vient vraiment à manquer. Le cache se remplit de nouveau dès qu'une requête réclame un objet qui n'est plus dedans.

Conclusion

S'il existe aussi des moyens de paramétrer le ramasse miette au niveau de la JVM grâce à la ligne de commande, l'objectif de cet article était de présenter des solutions d'inter action au niveau de la conception du logiciel. Nous pourrions développer ce sujet par quelques exemples et aussi par l'analyse de classes complémentaires telle que « WeakHashMap ». Ceci fera l'objet d'un autre article. Affaire à suivre.