Préprocesseur et Java

Michel CASABIANCA
casa@sweetohm.net

Les connaisseurs doivent se dire : "Ca y est ! Il a pété une durite Casa ! Il n'y a pas de préprocesseur en Java". Ce n'est pas faux, et cela peut d'ailleurs paraître une hérésie pour les fanatiques du C/C++. Cependant, il y a moyen de s'en passer pour au moins deux points :

La compilation conditionnelle

La compilation conditionnelle consiste, comme chacun sait, à ne compiler certaines portions du code que si une condition est réalisée. On utilise en C les directives de compilation #IF..#ENDIF. Le préprocesseur lit le fichier source avant la compilation et manipule le fichier source de manière à satisfaire à ces directives. Une utilisation de cette compilation conditionnelle est la réalisation des différentes versions d'un même logiciel avec un même et unique fichier source.

Imaginons que vous écriviez un shareware [^1] : vous souhaitez réaliser deux versions, l'une que vous diffuserez à grande échelle et qui se bloquera au bout de 30 jours d'utilisation, et une autre version complète pour les utilisateurs enregistrés. Il serait très pénible de devoir réaliser deux versions avec deux sources différents, car des modifications réalisées sur un source doivent être répercutées sur le second. La solution consiste à utiliser la compilation conditionnelle de manière à pouvoir compiler les deux versions avec le même source. Une autre utilisation est l'élimination de portions de code destinées au débuggage lors de la compilation de la version finale du projet.

On peut réaliser cette compilation conditionnelle avec un peu d'astuce, examinons le source suivant :

```
public class Preprocesseur
{
  static final boolean DEBUG=false;

  public static void main(String[] args) {
    System.out.print("Le programme a été compilé en ");
    if(DEBUG)
      System.out.println("mode débuggage.");
    else
      System.out.println("mode version finale.");
  }
}
```

On remarque tout d'abord que l'on définit une variable `static final boolean BUG`. Cette variable est static donc c'est une variable de classe, commune à toutes les instances d'une même classe. D'autre part, elle est final ce qui veut dire que l'on ne peut modifier sa valeur en cours d'exécution. Cette variable peut donc être assimilée à une constante, et le compilateur le sait, et va en tenir compte lors de la compilation en éliminant le test et la portion de code qui n'est jamais exécutée. On peut le vérifier facilement avec un éditeur hexadécimal : suivant que ce programme a

été compilé avec DEBUG sur true ou false, on trouve les chaînes "débuggage" ou "version finale".

Variables et méthodes "inlinées"

En C, on peut définir des constantes avec la directive define : #DEFINE PI=3.141592 par exemple. Ce qui peut paraître étrange à première vue, c'est qu'on ne donne pas le type de cette constante. On comprend que ce n'est pas nécessaire lorsqu'on sait que le préprocesseur agit avant la compilation en remplaçant les "PI" par "3.141592" dans le code source. Cette méthode de définition des constantes permet de gagner du temps à la compilation car le processeur n'a pas à aller chercher la valeur de PI, on lui fournit directement comme constante.

Il est possible de procéder de même avec Java, en déclarant une variable static final : le compilateur sait alors que c'est une constante (on ne peut pas lui allouer une autre valeur que celle donnée lors de la compilation), donc il remplace les références à la variable PI par sa valeur. On peut donc encore se passer du préprocesseur dans ce cas.

Dans le même ordre d'idée, on peut aussi parler du cas des méthodes déclarées final, static ou private. Ces méthodes ne pouvant être surchargées par un objet qui en hérite, le compilateur peut les inliner (c'est à dire les inclure dans le code à la place de l'appel à ces méthodes). Si une méthode n'est pas déclarée final, static ou private, le compilateur ne peut le faire car lors de l'appel à cette méthode, la machine virtuelle doit appeler la version surchargée et non celle de la classe parent. On peut donc optimiser son code Java en déclarant les méthodes que l'on est sur de ne pas surcharger comme final.

[^1]: Cet article date de 1996, époque à laquelle je ne connaissais pas GPL :o)