Chargement de classes en Java Application à OSGi™

Gaël Thomas
gael.thomas@lip6.fr

Université Pierre et Marie Curie
Master Informatique
M2 – Spécialité SAR

Pourquoi un chargeur de classes

Un chargeur de classe charge de nouvelles classes
dynamiquement dans la machine virtuelle

Évite de charger toutes les classes au démarrage
- ✓ Optimise le nombre de classes chargées
  Exemple : les classes d'exceptions ne sont chargées que si elles sont utilisées
- ✓ Permet de choisir dynamiquement les classes chargées en fonction du contexte
  Exemple : la classe d'encodage des caractères chargées dépend de l'alphabet utilisé sur la machine locale
- ✓ Permet de construire des serveurs d'applications

Pourquoi un chargeur de classes

Pourquoi définir de nouveaux chargeurs de classes

Pour décharger et mettre à jour des classes à l'exécution
- ✓ OSGi™, JEE, Web Component, CCM, Corba, RMI…

Pour chercher des classes absentes du CLASSPATH
- ✓ URLClassLoader, AppletClassLoader…

Pour modifier le bytecode des classes au chargement
- ✓ Décryptage, décompactage (jar files…)
- ✓ Insertion d'aspects (JAC, JBoss AOP…)
- ✓ Manipulation direct de bytecode (Javassist, Bcel, asm…)

Pour vérifier le bytecode
- ✓ Signature, vérification de type, sécurité…

Fonctionnement des chargeurs de classes

Un chargeur de classe charge une classe à partir de son nom ***de manière unique***
- ✓ Gère le chargement d'un ensemble de classes
- ✓ Gère les packages associés
- ✓ Gère l'accès aux ressources (images etc…) associés aux classes

Un chargeur de classe est une instance d'une classe qui hérite de la classe java.lang.ClassLoader

Un chargeur de classe possède un chargeur parent sauf le chargeur initial (qui charge rt.jar/core.jar)

Deux méthodes principales :
- ✓ loadClass(String) : cherche une classe parmi les classes déjà chargées
- ✓ findClass(String) : construit une nouvelle classe en mémoire

Fonctionnement des chargeurs de classes

Tout chargement est effectué dans le même chargeur de la classe qui initie le chargement, ***sauf*** avec les méthodes

- ✓ ClassLoader.loadClass(String name, boolean resolve)
- ✓ Class.forName(String name, boolean initialise, ClassLoader parent)

Remarques

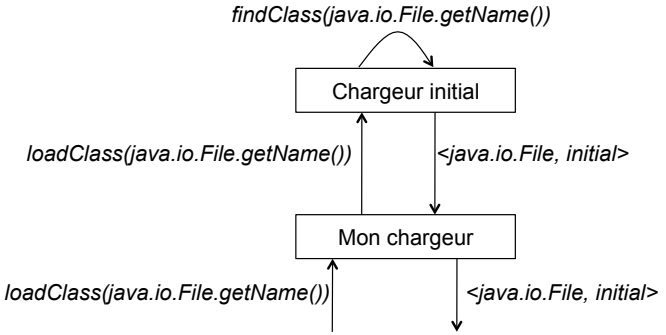
- ✓ Une classe possède un chargeur de classes et ne peut pas en changer
- ✓ Une classe est identifiée de manière unique par **son nom et son chargeur**
- ✓ Une même classe **peut être chargée dans plusieurs chargeurs** de classes

Fonctionnement par défaut :

- ✓ loadClass : cherche d'abord dans le chargeur parent
  Évite de charger plusieurs fois java.lang.Object
- ✓ findClass : cherche dans le CLASSPATH uniquement (loader applicatif)

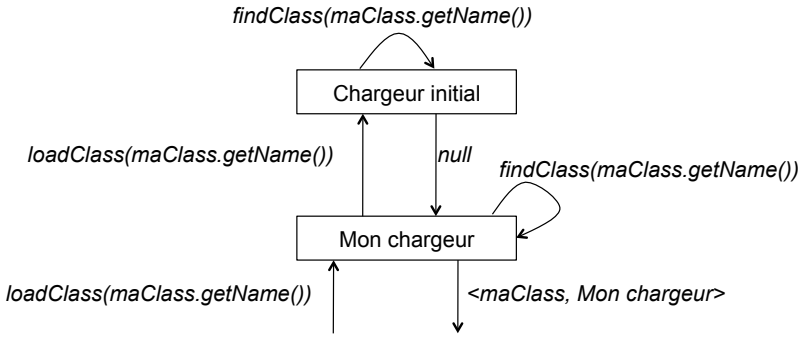
Fonctionnement des chargeurs de classes

Délégation par héritage : délégation vers le parent réussi



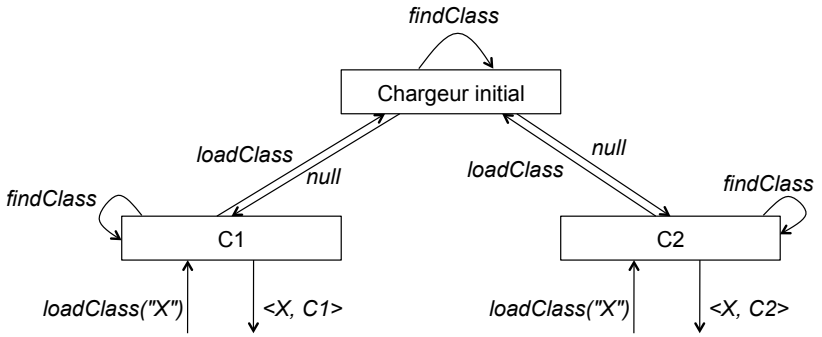
Fonctionnement des chargeurs de classes

Délégation par héritage : délégation vers le parent ratée



Fonctionnement des chargeurs de classes

Délégation par héritage : chargement multiple



<X, C1> et <X, C2> sont deux classes différentes

Peuvent avoir des bytecodes identiques, mais pas nécessairement!

Fonctionnement des chargeurs de classes

Arbre de délégation au démarrage

ClassLoader Bootstrap ou initial (natif et intégré à la VM)
- ✓ Charge les classes de démarrage (rt.jar/core.jar/glibj.jar)
- ✓ Pas de vérification de bytecode au chargement (rt.jar distribué avec la JVM)
- ✓ Sa référence est null

sun.misc.Launcher$ExtClassLoader (extension) (écrit en Java)
- ✓ Charge les classes des jarfiles présents dans le répertoire des extensions standards
- ✓ Pas de vérification de bytecode au chargement
- ✓ Son parent est le ClassLoader initial

sun.misc.Launcher$AppClassLoader (application ou système) (écrit en Java)
- ✓ Charge les classes et les jarfiles du CLASSPATH
- ✓ Vérification du bytecode au chargement
- ✓ Sa référence est donnée par ClassLoader.getSystemClassLoader()
- ✓ Son parent est le ExtClassLoader

Déchargement de classe

Un objet n'est libéré que si il n'est plus référencé
⇒ Une classe n'est libérée que si elle n'est plus référencée

Un chargeur de classe possède une table de hash de toutes les classes qu'il a chargé
⇒ Une classe n'est libérée que si son chargeur est libéré

Une classe peut être libérée pourvu que son chargeur ne soit plus référencé

⇒ Une classe peut être mise à jour dynamiquement

Déchargement de classe

Mise en œuvre difficile

- Un objet référence sa classe
- La classe référence son chargeur de classe
- Le chargeur de classe référence toute les classes qu'il a chargé

Pour libérer une classe il faut s'assurer qu'il n'existe plus aucune référence vers une des instances d'une des classes gérées par le chargeur de classe

Application à OSGi

Pour mettre à jour un bundle, il faut que les utilisateurs des services du bundle ne référencent plus les services fournis par le bundle…

⇒ prendre en compte les événements

Exemple de chargeur de classes

Le chargeur de classe (1/2)

```
public class Test extends ClassLoader {
  private String name;

  public Test(String name) { this.name = name; }
  public void finalize() {
    System.out.println("Destroy all classes"); }
  public String toString() {
    return "ClassLoader<" + name + ">"; }

  public static void main(String args[]) throws Exception {
    (new Test("one")).
      loadClass("Titi").getDeclaredMethod("f").invoke(null);
    (new Test("two")).
      loadClass("Titi").getDeclaredMethod("f").invoke(null);

     System.gc(); }
```

Exemple de chargeur de classes

Le chargeur de classe (2/2)

```
public Class findClass(String name) throws … {
  try {
    File file = new File("titi", name + ".class");
    InputStream is = new FileInputStream(file);
    int length = (int)file.length();
    byte b[] = new byte[length];
    is.read(b);
    // We can modify or verify the bytecode from b
    return defineClass(name, b, 0, length);
  } catch(Exception e) {
    throw new ClassNotFoundException(name);
  }
}
```


Exemple de chargeur de classes

La classe chargée notre nouveau chargeur

```
public class Titi {
  public static int a = 0;  // champs static : 1 par loader

  public static void f() {
    System.out.println("In f, a is " + a++);
    new Toto();   // chargement d'une sous-classe
  } }
```


Exemple de chargeur de classes

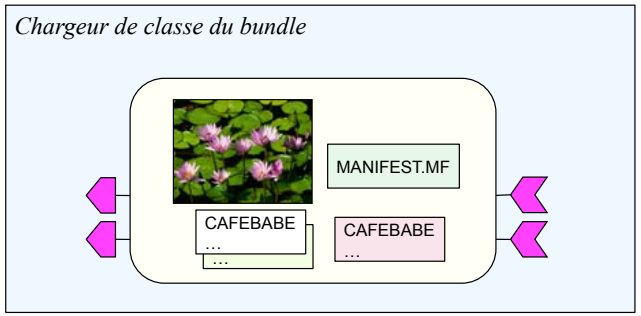
La classe chargée explicitement dans notre chargeur

```
class Toto {
  public Toto() {
    System.out.println("-- ClassLoader tree --");

    for(ClassLoader  cur=getClass().getClassLoader();
                     cur!=null; cur=cur.getParent())
      System.out.println("    " + cur);

    System.out.println("    primordial");
  }
}
```


Chargement de classes en OSGi™

L'unité de déploiement OSGi™ est le bundle

Un ClassLoader par bundle



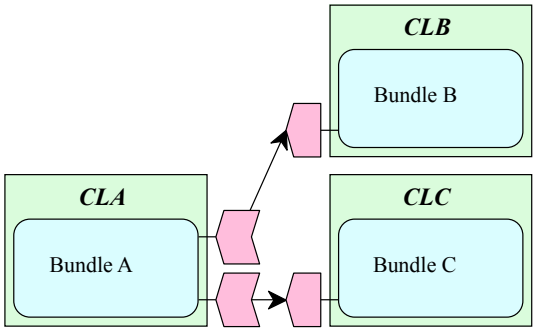
Chargement de classes en OSGi™

Un bundle est déchargé si et seulement si
il n'existe plus de référence vers une instance d'une des classes du bundle

- ✓ Plus de référence du framework vers le bundle
  Le bundle doit passer dans l'état uninstalled
- ✓ Plus de référence des autres bundles vers le bundle
  Les utilisateurs de services doivent relâcher les références vers les services

Chargement de classes en OSGi™

La délégation par héritage n'est pas suffisante
Preuve par l'absurde en utilisant la délégation naturelle Java



A utilise une classe de B
=> CLA fils de CLB
A utilise une classe de C
=> CLA fils de CLB

Or un chargeur ne possède qu'un unique parent…

Chargement de classes en OSGi™

Il existe de nombreux algorithmes de chargement de classes
- ✓ Par héritage (chargement naturel en Java)
- ✓ Par import/export (OSGi™)
- ✓ Par recherche exhaustive dans tous les chargeurs

Délégation Java non suffisante

Le Module Loader charge des modules
Abstraction des notions de packages, classes ou systèmes de fichiers

Le Module Loader est un chargeur de classes configurable
- ✓ La délégation est implantée par une politique
  Une implémentation de l'interface SearchPolicy
- ✓ Le chargement est laissé libre à l'utilisateur
  Une implémentation de l'interface ResourceSource

Chargement de classes en OSGi™

Un module est un bundle

La délégation utilise les packages exportés/importés
Les packages ont des versions
- ✓ Exportation avec un numéro de version
- ✓ Importation avec des ensembles de numéros de version

Délégation en OSGi
1. politique par défaut (rt.jar, ext.jar, classpath)
2. ni importé par A ou ni exporté ailleurs => recherche dans A
3. importé par bundle A => délégation au (premier arrivé) ML qui exporte
4. exporté par A mais déjà exporté ailleurs => le premier
5. Erreur : impossible de trouver une classe

Conclusion

Le déchargement de classes reste un problème difficile

- ✓ Nécessite une programmation (trop?) rigoureuse
- ✓ Éviter les références entre bundle Supprimer ces références lors des mises à jour
- ✓ Utiliser au maximum ServiceBinder/SCR
- ✓ Nécessite de redémarrer tous les bundles qui importent lors de la mise à jour de celui qui exporte

Toutefois, il n'existe aucun autre moyen pour mettre à jour des classes sans sortir des spécifications Java…

Pour quelle raison le problème ne se pose pas avec les EJB?
(Même avec les LocalHome, Local)