Chargeurs de classes Java (ClassLoader)

Didier Donsez

Université Joseph Fourier - Grenoble 1

PolyTech'Grenoble - LIG/ADELE

Didier.Donsez@imag.fr

Didier.Donsez@ieee.fr

Licence

- Cette présentation est couverte par le contrat Creative Commons By NC ND
  - http://creativecommons.org/licenses/by-nc-nd/2.0/fr/

Kequoi ca un chargeur de classes ?

- Son rôle est

1) de charger le bytecode d'une classe depuis un artéfact (archive Java, répertoire distant …)

2) communiquer le bytecode à la machine virtuelle

Pourquoi utiliser les chargeurs de classes

- Classes non présentes dans le CLASSPATH
  - URLClassLoader, AppletClassLoader, …
    - ex: WEB-INF/classes et WEB-INF/lib d'une WebApp
    - ex: CODEBASE d'une applet, …
- Déchargement et Mise à jour du bytecode lors de l'exécution de la VM (runtime)
  - Chargeurs de OSGi
- Modification du ByteCode à la volée au chargement
  - Instrumentation
    - AOP (Aspect Oriented Programming)
    - BCEL, ASM
  - Protection
- Chargement de ressources associées à la classe
  - properties, images, …
- Recherche de Service Providers ou de Drivers
  - META-INF/services (java.util.ServiceLoader de 6.0)

Principe de la délégation (Java 2)

- Tout chargeur a un chargeur parent
  - sauf le chargeur primordial
- Tout chargeur vérifie si la classe à charger n'a pas déjà été chargée par un chargeur parent

- Remarque
  - Toute classe est attachée à un chargeur et ses instances ne peuvent en changer
  - Un même espace de nom (classe) peut être chargé de manière séparé par plusieurs classes loaders
    - même ou différentes versions

Chargeur et Bibliothèque native

- Classe comportant des méthodes natives
- La bibliothèque dynamique (.dll,.so) doit être chargée en mémoire virtuelle
  - à l'exécution du code statique d'initialisation
    - Juste après le chargement/vérification
  - lors de l'appel de la méthode « wrapper »
- Exemple

```
class NativeTest {
private static islibloaded=false;
private native String nativefnc(String param); // JNI
private void loadlib() throws throws SecurityException, UnsatisfiedLinkError {
    System.loadLibrary("MYLIB");
    islibloaded=true;       }
public String fnc(String param) {
    if(!islibloaded) loadlib();
    return fnc(param);      }           }
```

Arbre de délégation

- ClassLoader bootstrap ou primordial
  - Charge les classes de boot (rt.jar)
  - Natif (C) et intégré à la VM
  - Pas de Vérification du ByteCode au chargement
  - Sa référence est null
- sun.misc.Launcher$ExtClassLoader *(extension)*
  - Charge les classes des jarfiles présents dans le répertoire des extensions standards
  - Pas de Vérification du ByteCode au chargement
  - Son parent dans l'arbre de délégation est le CL primordial
  - Écrit en Java
- sun.misc.Launcher$AppClassLoader *(application ou system)*
  - Charge les classes des répertoires/jarfiles du CLASSPATH
  - Vérification du ByteCode au chargement
  - Sa référence est donnée par ClassLoader.getSystemClassLoader()
  - Son parent dans l'arbre de délégation est ExtClassLoader
  - Écrit en Java

Arbre de délégation

- Affichage de l'arbre de délégation

```
ClassLoader loader=getClass().getClassLoader();
System.out.println("ClassLoader delegation tree");
ClassLoader traceloader=loader;
for(int i=0;;i--) {
    if(traceloader==null) {
        System.out.println("classloader("+i+")=primordial");
        break;
    } else {
        System.out.println("classloader("+i+")="+traceloader);
        traceloader=traceloader.getParent();
    }
}
```

```
classloader(0)=donsez.CryptoClassLoader@2152e6
classloader(-1)=sun.misc.Launcher$AppClassLoader@bac748
classloader(-2)=sun.misc.Launcher$ExtClassLoader@7172ea
classloader(-3)=primordial
```

classes chargées par des chargeurs différents

- Cas de l'interface
- Cas de la classe
- Voir livre Halloway

Chargeur et Threads

- ClassLoader Thread.getContextClassLoader()
  - Par défaut, celui de la thread parent
  - La thread primordial utilise le chargeur de l'application
- Thread.setContextClassLoader(ClassLoader)

-Xbootclasspath

- Permet le remplacement de tout ou partie des classes du JRE chargées par le CL bootstrap
- Exemple

```
javac –d .\boot  myrtsrc\java\kang\Integer.java

java –Xbootclasspath:.\boot;c:\jdk1.2.2\jre\lib\rt.jar –classpath . MyIntegerTest
```

Membres statiques et ClassLoader

- Chargement d'une classe par des CL différents

```
class Compteur {
    static int cpt;
    {
      cpt=0;
      incr();
    }
    static int incr(){
      cpt++;
      System.out.println("Nlle valeur: "+cpt);
      return cpt;
    }
}
```

- Une seule solution
  - Faire charger la classe par le CL primordial

Bibliothèques de code natif (JNI)

Signed Class/Jar

- Tools : JarSigner (+KeyTool)
  - Sign and verify entry by entry (class by class)
  - Use the local keystore
  - Several « co-signers »
- Use for applet sandoxing (extended permissions)
- SecurityManager
  - Signature is checked at loading
  - And use in permission

- See Also
  - Courses on JCE (in french)

Protected ByteCode

- Motivation
  - Software licencing
    - Quelques chiffres
      - Utilisation illégale de logiciel
        - 50% en Europe
        - 95% en Asie, Amerique latine, Europe de l'Est
      - Perte de revenu pour les développeurs
        - $12 milliards par an sur le monde entier
        - $3 milliards par an aux USA seul.
- DongleClassLoader
  - Classes are encrypted with one key per licence
  - Decryption is done by a Dongle (or SmartCard) containing a unique decryption key
  - The presence of the dongle is also checked regulary

Propriétés

- java.class.path
  - Le « CLASSPATH »
- java.ext.dirs
  - Répertoire contenant les jarfiles chargées par le ExtClassLoader
- sun.boot.class.path
  - Liste des répertoires/jarfile contenant les classes chargées par le CL bootstrap
- java.library.path
  - Liste de répertoires contenant les bibliothèques (de fonctions natives) à charger

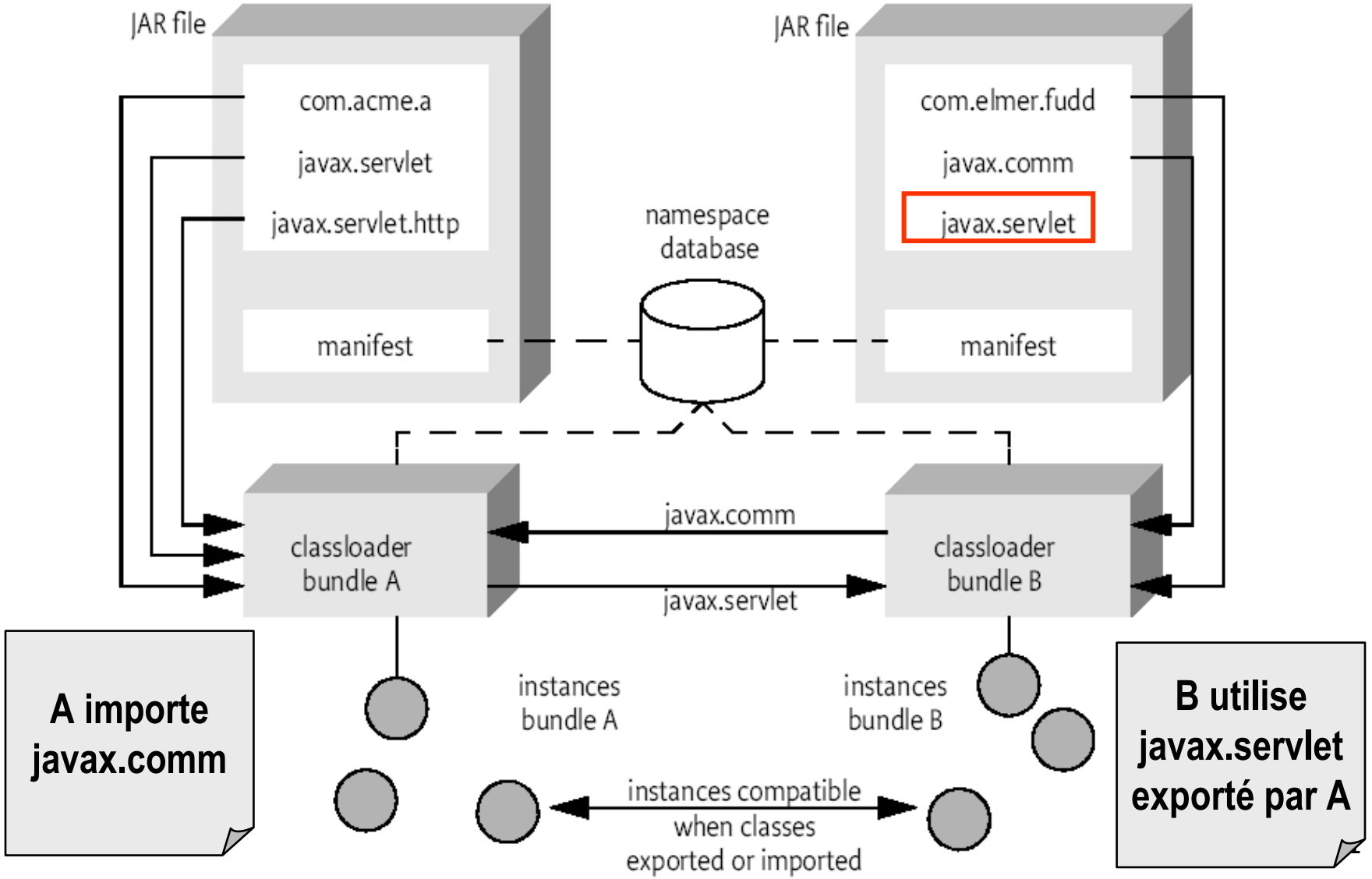
ClassLoader et Applet

- Chargement classe par classe
- Chargement groupé (.jar)

ClassLoader et OSGi

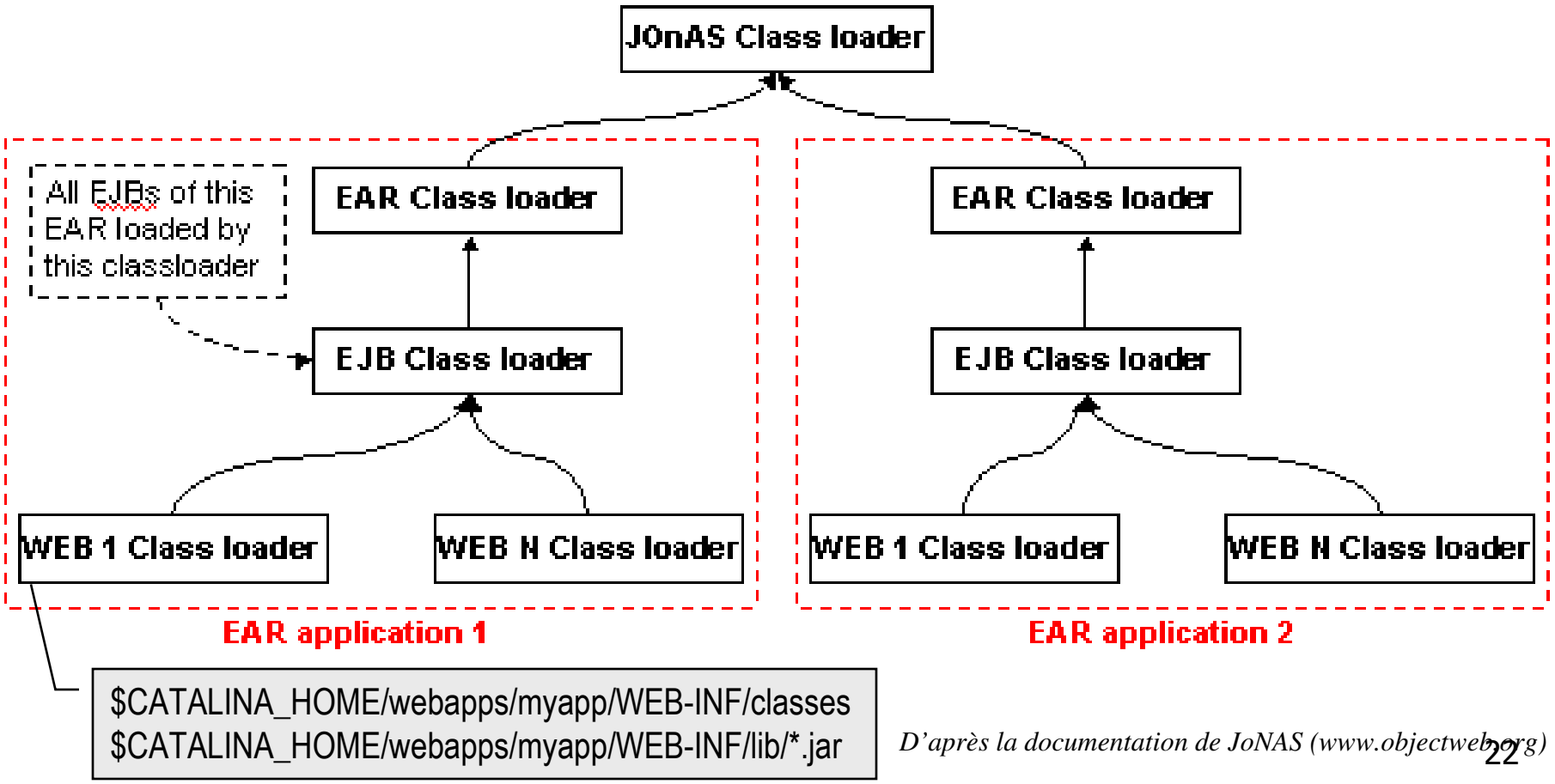
- Orienté vers le déploiement de code sur des passerelles résidentielles (systèmes embarqués)
- Bundle
  - Unité de déploiement du code (et des services)
  - Tout concepteur d'application est gagnant à distribuer son application comme un ensemble de bundles
    - évite le casse tête du CLASSPATH
- 1 ClassLoader par Bundle
  - Chargement, Mise à Jour, Déchargement
- Règle de délégation
  - BundleClassLoader contacte ses « voisins » pour charger les classes des packages *versionnés* listés dans « Import-Package »

ClassLoader et OSGi



ClassLoader et J2EE

- Web Application + Enterprise Application



D'après la documentation de JoNAS (www.objectweb.org)

Etude de Cas : WAR et OSGi

- Motivations
  - Plugins BIRT Eclipse utilisés
    dans des servlets déployés avec Tomcat
  - DysoWeb : mini-webapps chargées/déchargées dynamiquement
- Principe
  - Embarquer un canevas OSGi dans la WebApp
  - Le CL du canevas OSGi est celui de la WebApp
  - Les CL des plugins
  - TODO

Android + .dex file format

RMIClassLoader

- RMI, JINI
- TODO

JNLP

- TODO

ANT ClassLoader

- org.apache.tools.ant.AntClassLoader
- org.apache.tools.ant.loader.AntClassLoader2
- org.apache.tools.ant.util.ClasspathUtils

ClassLoader et J2ME/CDC/PBP

- Chaque Xlet doit être chargé dans un classloader séparé

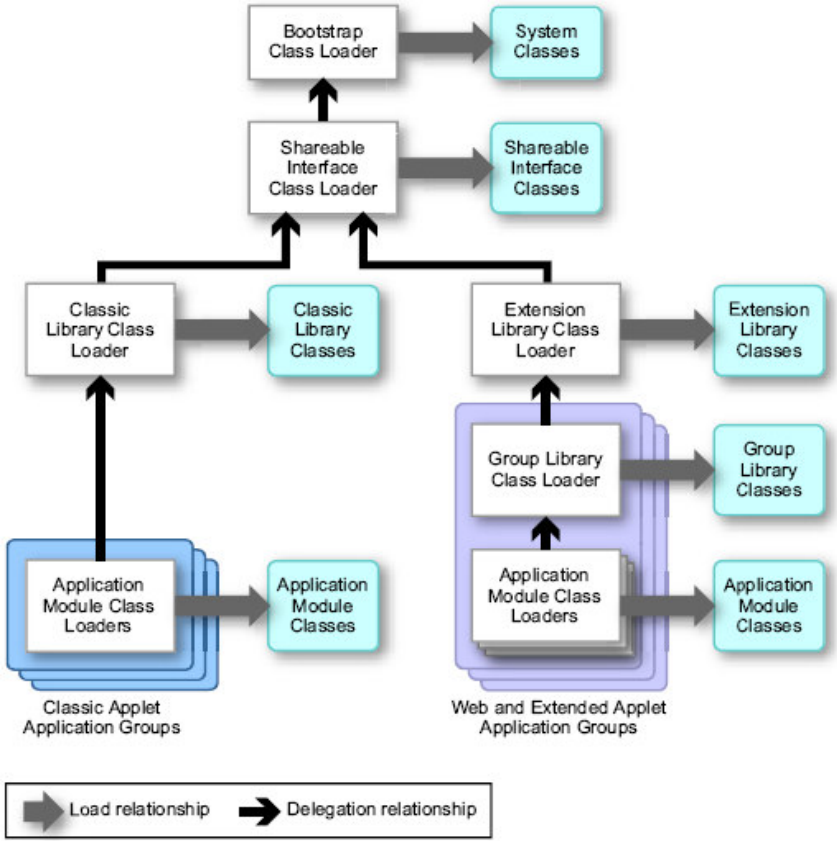
ClassLoader et embarqué

- J2ME/CDLC(KVM)
  - Bootstrap ClassLoader
    - intégré dans de code (C) de la VM
  - Pas de délégation (pour l'instant)
- TINI
  - Édition de lien anticipée (format .tini)
  - Chargement de classes dynamique possible (Class.forName() mais occupe plus de mémoire)
- JavaCard
  - format « Just-In-Place » .cap : peu ou pas de transformation pour être utilisé par la JCVM
- ROMIFICATION
  - Fermeture des chargements de classes
- IBM J9 *http://www-306.ibm.com/software/wireless/wece*
  - Format JXE : format utilisé par la J9 pour les JAR installés en ROM ou en FlashRAM
    pour l'exécution « in-place »

- Visiter le source (C) de la WabaVM

ClassLoading in Java Card Platform, v3.0, Connected Edition

- JC3.0 Connected Edition (http://java.sun.com/products/javacard/3.0) enables code isolation thru ClassLoaders
- ClassLoader delegation principle but no user-defined class loaders



ModuleLoader (Rick Hall)

- Module
  - Set of classes/interfaces and ressources
- High level requirement
  - Dynamic (at runtime)
    - Load
      - Transformation
      - Licencing
    - Update
    - Replace
    - Remove (non stop VM)
      - Event
  - Versioning
    - Multi namespace
    - 2 versions running on the same JVM
  - Secure
  - Generic/neutral
    - Deployment
    - Communication
  - Native Libraries
- Implemented in Oscar and Felix
- See *Richard S. Hall: A Policy-Driven Class Loader to Support Deployment in Extensible Frameworks. Component Deployment 2004: 81-96*

JSR 277 et JSR 294

- TODO
- JSR 277 Java Module System

- Jigsaw project

- See
  - H2K : a preliminary implementation of JSR277

Appication Hot Redeployment

- Motivation
  - limit the unavailabilty time of applications
  - In Web frameworks, Application servers, …
- Solution : on-the-fly class reloading
  - Possible only if members and method signatures are unchanged
  - Case of method bug fixes
- Framework
  - Java 1.4 HotSwap
  - JavaRebel http://www.zeroturnaround.com/javarebel/
    - JVM Agent

Et .NET

- TODO
- Notions
  - Application domain
  - Assembly

Références

- S. Liang and G. Bracha. Dynamic class loading in the Java Virtual Machine. In ACM Symp. on Object-Oriented Programming: Systems, Languages and Applications 1998, volume 33(10) of Sigplan Notices, pages 36--44. ACM Press, October 1998
- Halloway, Stuart. Component Development for the Java™ Platform, Publ. Addison Wesley Professional, 2002, ISBN: 0-201-75306-5,
  - version PDF téléchargeable du livre sur http://staff.develop.com/halloway/compsvcs
- Neward, Ted.*Javageeks.com White Papers*. 1999-2001. http://www.javageeks.com/Papers
  - Plusieurs papiers et codes sur les chargeurs de classes.
- Gabriel Bizzotto, « JITS, Java in the Small », Rapport de DEA, LIFL, Juin 2002
  - Étude sur les chargeurs de classes orienté vers les petites devices
  - http://www.lifl.fr