Plate-forme J2EE open source : JONAS 3.2

(V. Alvaro 09/2003)

Introduction

JonAS est serveur d'application Open Source de ObjectWeb qui implémente l'ensemble des spécifications J2EE, et supporte les services Web. ObjectWeb est un consortium Open Source sans but lucratif, sur le modèle de la fondation Apache. Son objectif est de fournir un ensemble de logiciels middleware en Open Source. Il regroupe notamment l'INRIA, Bull, France Télécom R&D...
Jonas fournit :

- Un conteneur D'EJB (Stateless et Stateful session Beans, Entity Beans avec Bean-Managed ou Container-Managed Persistance (CMP), Message-Driven Bean, interfaces locales, etc…
- Un conteneur web pour les applications J2EE (Tomcat et Jetty)
- Des transactions distribuées
- Scalability
- Support de sécurité
- Gestion basée sur JMX
- Intégration JMS
- Services utilisateurs et intégrés pour une configuration et une adaptation du serveur plus facile.

- Le support des services Web, via l'intégration d'Axis (la conformité J2EE 1.4 est également annoncée)
- Une intégration renforcée avec les environnements de développement Eclipse, JBuilder, et XDoclet.

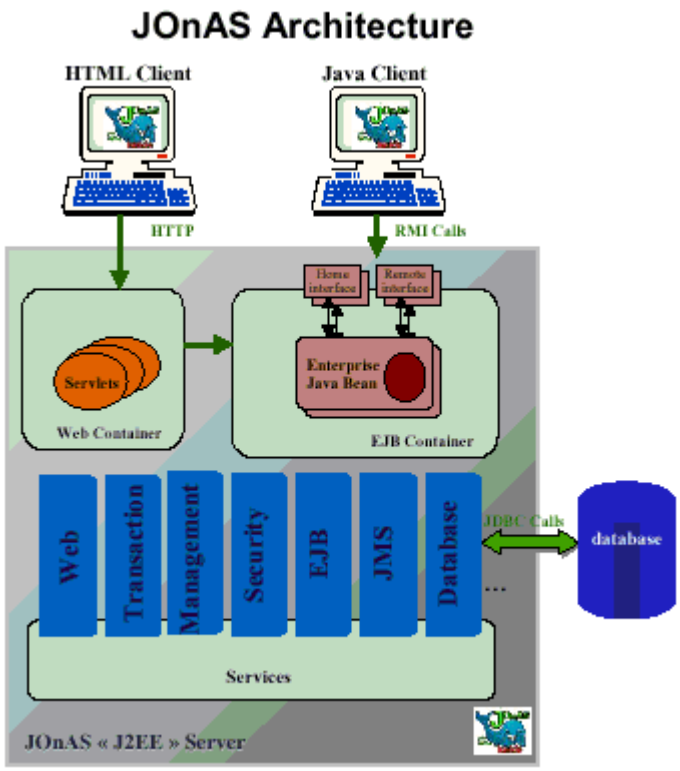
Jonas a déjà été utilisé :

- sur différents OS : Windows*, Linux, Solaris, AIX, HP-UX, Novell, etc.
- sur différentes JVM
- sur différentes base de données : Oracle, PostgreSQL, MySQL, SQL server, Access, DB2, ObjectStore, Informix, Interbase, Versant,etc.

Jonas a été utilisé pour développer des projets à grande échelle de : e-commerce, e-portail, système de management, application intranet, processing de documents, systèmes d'inventaire, applications bancaires, etc…

Architecture de Jonas

Pour avoir une vue globale, on peut dire que Jonas s'appuie sur Tomcat ou Jetty pour son conteneur WEB, et sur des composants du consortium ObjectWeb pour son ORB (Jonathan), pour JMS (JORAM), et pour la persistance CMP2.0 (JORM). Il fournit par défaut des mécanismes permettant une bonne tenue en charge, ainsi que des solutions de clustering. Il s'administre à travers une console qui est elle-même une application web, s'appuyant sur JMX. Il supporte le déploiement de connecteurs JCA, permettant l'accès aux systèmes d'entreprise propriétaires.



Services et spécifications

Jonas est construit en termes de services (qui supportent les spécifications indiquées) :

- EJB 1.1 : les conteneurs de EJB sont fournis par un set de classes Java et un outil qui génère des classes d'interposition. Jonas dépend des projets JORM et MEDOR d'ObjectWeb pour implémenter CMP2.0
- JTA 1.0.1/JTM : le Transaction Manager fournis des API Java support de transaction et de coordination de transaction distribuée.
- JDBC 2.0 : un Manager de Database qui fournit un support de Connectivité aux Bases de données.
- JMS 2.0.1 : un service Java Messaging fournit par JORAM (implémentation de JMS d'ObjectWeb), une technologie venant de ScalAgent. Jonas peut aussi supporter d'autres implémentation de JMS.
- JMX : Jonas contient une technologie Java Management Extension. La gestion et le monitoring sont accessible via une interface web (servlet basée sur GUI).
- JCA 1.0 : Jonas supporte l'architecture J2EE Connector qui définit une série de mécanismes permettant l'intégration d'EIS (Enterprise Information Systèmes).

(A *Connector Resource Service* for deploying JCA-compliant Ressources Adapters)

- Un service de Gestion de la sécurité.
- Un Web Container Service pour faire tourner un conteneur de serlet/jsp (Tomcat ou Jetty) comme un service de Jonas, dans la même JVM.
- Un service EAR pour déployer les applications archivées en fichiers EAR
- Une série d'outils pour concevoir, déployer et gérer des EJB.
- Un service Mail qui fournit des composants d'application avec les ressources nécessaires pour envoyer des mails.

Les EJB

Définition

Les EJB (Enterprise JavaBeans) sont des composants Java portables, réutilisables et déployables qui peuvent être assemblés pour créer des applications. Tous ce qui concerne le domaine métier est pris en charge dans le tiers métier par des EJB. Ils s'exécutent dans un conteneur EJB qui va leur fournir des services tels que les transactions ou la persistance...
Ces composants ont plusieurs avantages :
- Le conteneur EJB fournit aux Enterprise JavaBeans des services.
Le conteneur va être responsable de fournir aux EJB des services tels que la gestion des transactions, des autorisations, de la persistance... Le développeur n'ayant plus à s'occuper de ces choses, il peut se concentrer sur le développement de son EJB.
- Les EJB contiennent la logique applicative.
Les applications clientes n'ont plus de routines contenant les règles métiers ou les accès bases de données. Elles sont donc plus légères et plus souples.
- Les EJB sont portables.
Une application peut être construire à partir d'EJB existants et être déployée sur n'importe quel serveur compatible J2EE.

Un EJB reçoit les données du client , les traite si nécessaire, et les envoies vers le tiers système d'information de l'entreprise pour stockage. Il peut aussi faire la manœuvre inverse. Il existe 3 EJB : session beans, entity beans, et message-driven beans.

- session bean : représente une conversation transitoire avec le client. Quand le client fini l'exécution, le bean et les données disparaissent.
- Entity bean : représente une données persistante qui est stockée dans une ligne de table de base de données. Quand le client se déconnecte ou si le serveur s'éteint, les services sous jacents se chargent de sauvegarder l'entity bean.
- Le message-driven bean combine les caractéristiques du session bean et de l'écouteur de message du JMS (Java Message Service), qui permet à un composant métier de recevoir un message JMS asynchrone.

(Les EJB de type "Session" : Ils exécutent une tâche pour le client.
Les EJB de type "Entity" : Ils représentent un objet métier qui existe dans le système de stockage permanent (par exemple : un client ou une facture).
Les EJB de type "Message-Driven" : Ils permettent le traitement des messages asynchrones.)

Le conteneur EJB

Chaque instance d'un EJB se construit et vie dans un conteneur. Le conteneur est en fait un environnement d'exécution fournissant des services aux Entreprise JavaBeans.
Parmis ces services, on a par exemple :
- Les connexions à la base de données
- Les transactions
- La sécurité etc…

Les différents types d'EJB

- Les EJB de type session

Un EJB de type session est chargé d'effectuer une tâche pour un client. Dans cette catégorie d'Entreprise JavaBeans, il y a deux groupes :

- *Sans état :* Un composant de session sans état ne maintient pas d'état conversationnel. L'exemple typique est un convertisseur Euro/Franc qui aurait une seule méthode : euroToFranc(double valeur).
  Toutes les invocations de méthodes que vous ferez sur un EJB sans état que vous avez instancié ne seront pas forcément traitées par le même EJB.

- *Avec état* : Un composant avec état est dédié à un certain client pendant toute la durée de son instanciation. Concrètement, si vous modifiez une variable d'instance du composant, vous retrouverez cette valeur lors de vos prochains appels. Pour résumer, toutes les invocations d'une méthode par le client seront traitées par le même EJB.

→ Ce type d'EJB s'occupe des traitements.

- Les EJB de type Entity

Ce type d'EJB peut désigner quelque chose de concret (un client ou une facture) ou d'abstrait (une enchère ou une réservation). Sa caractéristique la plus importante est la persistance, c'est à dire que ce composant existe physiquement sur un support de stockage comme une base de données, un serveur LDAP ou un fichier XML.

La persistance modélisant les entity beans sont de deux types :
- *Bean-Managed Persistence*, où le fournisseur de bean doit développer la méthode d'accès à la base de données en utilisant l'interface JDBC.
- *Container-Managed Persistence*, dont l'accès à la base et le stockage sont automatiquement géré par l'environnement EJB. Le fournisseur n'a qu'à spécifier un descripteur de bean contenant les informations à propos du mapping des champs dans le schème de la base.

→ Ce type d'EJB s'occupe des données**.**

- Les EJB de type Message-Driven

Un Message-Driven Bean est un EJB qui va permettre à votre application de traiter des messages de manière asynchrone. En fait, cet EJB va réagir aux messages reçus au travers de JMS (Java Message Service)

→ Ce type d'EJB permet de traiter les données de manière asynchrone.

Cadre d'utilisation des EJB

Les EJB sont particulièrement recommandés :
- Lorsque l'application doit pouvoir gérer des montées en charge.
On peut facilement mettre en cluster les serveurs d'application.
- Lorsque l'application a besoin des transactions.
Les EJB supportent les transactions (peu importe le système de stockage utilisé).
- Lorsque l'application doit être accessible depuis de nombreux types de clients (
Applications, Sites web, PDA...).N'importe quel type de client pourra accéder aux EJB et pourra utiliser la logique et les données de l'application.
- Lorsque l'application est développée par plusieurs personnes.
Les EJB permettent de contenir, et donc de centraliser la logique applicative et ainsi d'assurer l'intégrité des données et le respect des règles métiers. Les développeurs des applications clientes peuvent se concentrer sur la présentation, il n'ont plus dans leurs codes de règles métiers ni d'accès à la base de données.

Rédaction d'un EJB

Le composant EJB se compose de 3 entités :
- L'interface locale (Interface Home) qui définit les méthodes qu'un client peut invoquer pour créer, trouver ou supprimer l'EJB . Elle est mise en oeuvre par le conteneur lors du déploiement dans une classe appelé EJB Home.

- L'interface distante (Interface Remote) qui contient les méthodes qu'un client de cet EJB peut appeler . Elle est mise en oeuvre par le conteneur lors du déploiement dans une classe appelée EJB Object. Le client du bean après avoir utilisé l'interface EJB Home pour accéder au bean, utilise cette interface pour appeler indirectement les méthodes métiers implémentées dans la classe bean.

- L'EJB qui est le composant (classe du Bean). La classe encapsule les données associées au bean et contient les méthodes métiers implémentées par le développeur qui permettent d'accéder à ces données. Elle renferme aussi les différentes méthodes utilisées par le conteneur pour gérer le cycle de vie de ses instances.

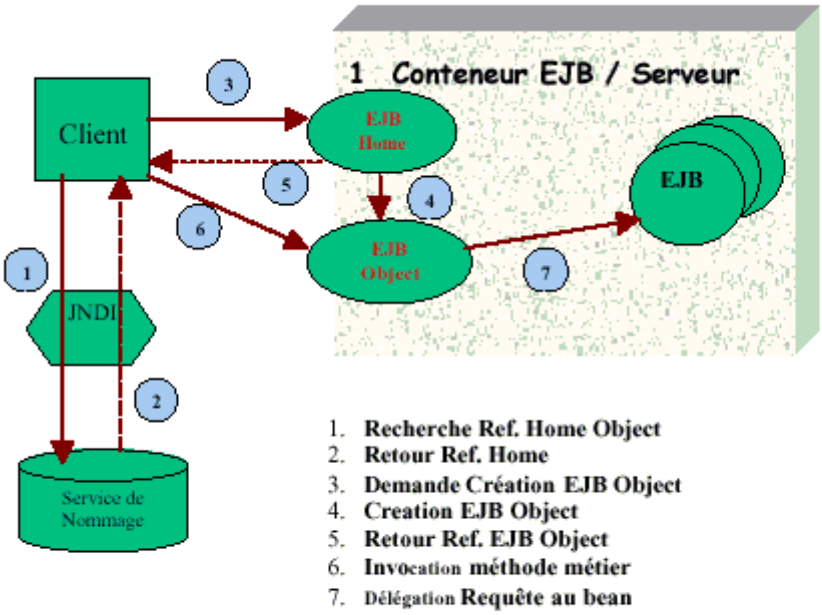


Figure 1 - Scénario relatif à l'appel d'une méthode bean par un client

Les trois fichiers qui suivent appartiendront au package sb (simple bean).

→ Interface locale (Interface Home)

Cette interface définit les méthodes utilisées par un client pour créer, rechercher ou supprimer des instances du bean. Elle est mise en oeuvre par le conteneur lors du déploiement dans une classe appelé EJB Home.

HelloWorldHome.java

```
package sb;
import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;
```

```
public interface HelloWorldHome extends EJBHome {
  HelloWorld create() throws RemoteException, CreateException;
}
```

Lorsqu'un client veut utiliser un EJB, il se sert de JNDI pour obtenir une référence à un objet qui est une instance de l'interface locale. Comme vous pouvez le constater, la méthode create() qui est appelée retourne un objet de type HelloWorld (type de notre interface distante).
Le client va donc pouvoir utiliser cette instance du même type que l'interface distante pour invoquer les méthodes de l'EJB.

→ L'interface distante (Interface Remote)

Cette interface va définir toutes les méthodes fonctionnelles (business method) qu'un client peut invoquer sur notre composant. (elle spécifie l'ensemble des fonctions qu'un client peut appeler).

Hello.java

```
package sb;
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface HelloWorld extends EJBObject {
   public String sayHelloWorld() throws RemoteException;
}
```

→ L'EJB

On code ensuite l'EJB pour implémenter les fonctions que l'on a "promis" au client dans l'interface distante, c'est à dire la fonction sayHello().

HelloWorldBean.java

```
package sb;
import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class HelloWorldBean implements SessionBean {

  public String sayHello() {
   return "Bonjour monde";
  }

  public void ejbCreate() {}
  public void ejbRemove() {}
  public void ejbActivate() {}
  public void ejbPassivate() {}
  public void setSessionContext(SessionContext sc) {}

}
```

L' EJB implémente l'interface SessionBean ce qui nous oblige à implémenter les interfaces suivantes :
- ejbActivate()

- ejbPassivate()
- ejbRemove()
- setSessionContext()

Ces méthodes sont invoquées par le conteneur pour notifier à votre EJB qu'un évènement s'est produit.
La méthode setSessionContext() est appelée immédiatement après l'instanciation du composant par le container. Elle reçoit un argument de type SessionContext que le composant peut utiliser pour faire appel aux services du conteneur.
Après setSessionContext() le container appelle la méthode ejbCreate().

Note : Pour chaque méthode create() définie dans son interface locale, le composant doit implémenter une méthode ejbCreate() correspondante, retournant void et contenant le même nombre et le même type d'arguments que la méthode create() source.

Création des descripteurs de déploiement de l'EJB

Pour un EJB, il y a deux descripteurs :
-Le descripteur de déploiement standard de sun : Le programmeur de l'EJB doit fournir avec son composant un descripteur de déploiement. Ce fichier décrit par exemple quelle classe est l'implémentation, l'interface locale et l'interface distante de l'EJB.
Ce fichier doit s'appeler ejb-jar.xml. (cf annexe 1)
- Le descripteur spécifique à JonAS : Certaines informations nécessaires au déploiement de l'EJB dans JOnAS ne sont pas dans la DTD précédente. Pour résoudre ce problème, il faut créer un autre descripteur de déploiement spécifique à JOnAS dont on peut trouver la DTD à l'adresse suivante : http://www.objectweb.org/jonas/dtds/jonas-ejb-jar_2_4.dtd.
Le fichier spécifie le nom JNDI de l'objet qui est l'interface Home du bean. C'est ce nom JNDI que les clients de notre application vont appeler pour accéder à l'interface locale de notre EJB.
Ce document devra s'appeler jonas-ejb-jar.xml. (cf annexe 1)

Attention : L'un des principaux avantages des EJB est leurs capacités à gérer des transactions d'une manière déclarative, et non pas codé dans l'application serveur, au moment de déploiement. C'est ce qu'on appelle le « Container-managed transaction demarcation ». Le comportement d'une transaction est défini au moment de la configuration, et dans le descripteur de déploiement du bean.

Packaging de l'EJB

L'EJB doit être packagé dans une archive JAR qui devra contenir :
- Les classes de l'EJB
- Les 2 descripteurs de déploiement (ejb-jar.xml et jonas-ejb-jar.xml).

Déploiement de l'EJB

Pour déployer l' EJB sous JOnAS, le fichier ejb-jar doit posséder les classes d'interposition l'EJB avec les services offerts par le Serveur d'EJB. Ces classes vont être crées à l'aide de l'utilitaire **GenIC** fourni avec JOnAS. Ensuite, pour déployer l' EJB dans JOnAS, il suffira de copier le fichier .jar vers le répertoire "C:\java\plateforme\jonas\ejbjars" et de modifier la configuration de JOnAS.

Cadres fonctionnels

Accès aux bases de données pour les EJB

L'accès aux bases de données passent par des sources de données de type **javax.sql.DataSource**. Ce composant (une factory) permet d'accéder à un objet de type **java.sql.Connection** permettant d'exécuter et de traiter des requêtes SQL. La déclaration de ces sources de données est faite dans le fichier %JONAS_ROOT%\jonas.properties.
…
jonas.datasources acces_1
…
Acces_1 représente un alias vers le fichier **acces_1.properties** contenant tous les paramètres d'accès à la base de données via JDBC :
Exemple de fichier **acces_1.properties** :
datasource.name **acces_1**
datasource.url jdbc:postgresql://hexadev/base1
datasource.classname postgresql.Driver
datasource.username hexadev
datasource.password vedaxeh

Un **entity bean** dans le cas **Bean-Managed Persistence** ou un **session bean** vont définir dans leur fichier de déploiement **ejb-jar.xml** l'accès à ces ressources.

Exemple dans un fichier de déploiement :
…
<jonas-resource>
<res-ref-name>jdbc/**uneBase**</res-ref-name>
<jndi-name>**acces_1**</jndi-name>
</jonas-resource>
…
L'accès à cette base se fait par JNDI**,**

Exemple d'utilisation :

InitialContext it = new InitialContext();
DataSource ds = (DataSource)it.lookup("java:comp/env/jdbc/**uneBase**");
Connection connection = ds.getConnection();
On utilisera de préférence la méthode **ejbCreate** pour obtenir une connexion et l'initialisation des différents champs mappant une partie de la base. Pour optimiser la communication client/serveur voir la fiche OPT_ETB_EJB_01.

Annexes

Bibliographie

- Toutes les docs d'utilisation de Jonas se trouvent sur : http://www.objectweb.org/jonas/doc/
- Les spécifications des EJB sont disponibles à l'adresse suivante ;
http://java.sun.com/products/ejb/docs.html
- Description de cadres fonctionnels pour les EJB :
http://www.djefer.com/articles/fiches/FD_01.html

Annexe 1 : fichiers de déploiement

ejb-jar.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
"http://java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb-jar>
  <description>Déscripteur de déploiement de HelloWorld</description>
  <display-name>HelloWorld</display-name>
  <enterprise-beans>
    <session>
      <description>Déscripteur de déploiement de HelloWorld</description>
      <display-name>HelloWorld</display-name>
      <ejb-name>HelloWorld</ejb-name>
      <home>sb.HelloWorldHome</home>
      <remote>sb.HelloWorld</remote>
      <ejb-class>sb.HelloWorldBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
  <assembly-descriptor>
    <container-transaction>
        <method>
          <ejb-name>HelloWorld</ejb-name>
          <method-name>*</method-name>
        </method>
        <trans-attribute>Required</trans-attribute>
    </container-transaction>
   </assembly-descriptor>
</ejb-jar>
```

jonas-ejb-jar.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE jonas-ejb-jar PUBLIC "-//ObjectWeb//DTD JOnAS 2.4//EN"
"http://www.objectweb.org/jonas/dtds/jonas-ejb-jar_2_4.dtd">
<jonas-ejb-jar>
  <jonas-session>
    <ejb-name>HelloWorld</ejb-name>
    <jndi-name>myHelloWorld</jndi-name>
  </jonas-session>
</jonas-ejb-jar>
```

Annexe 2 : aide pour différents aspects de Jonas.

1. http://www.objectweb.org/jonas/current/doc/Tools.html
2. http://www.objectweb.org/jonas/current/doc/Config.html
3. http://www.objectweb.org/jonas/current/doc/Config.html
4. http://www.objectweb.org/jonas/current/doc/Tools.html#jonasadmin

5. http://www.objectweb.org/jonas/current/doc/Tools.html#jonasadmin
6. http://www.objectweb.org/jonas/current/doc/Admin.html#Jadmin
7. http://www.objectweb.org/jonas/current/doc/Admin.html
8. http://www.objectweb.org/jonas/current/doc/PG_Entity.html#Tuning