

Leçon 4 – Client lourd

Afin de tester vos « session beans », vous avez déjà mis en oeuvre une application cliente (non web) simple. Dans cette leçon, nous explicitons les technologies mises en oeuvre dans le dialogue client-serveur.

1 PREMIÈRE APPROCHE

Dans la leçon précédente, vous avez exécuté votre application cliente au travers de Netbeans. Rien n'empêche de l'exécuter indépendamment. Comment ? Netbeans l'a construite aux normes « Java Web Start »¹ afin de pouvoir l'exécuter à partir d'une URL.

Exercice

Une fois votre application eVente déployée, connectez-vous via votre navigateur préféré à l'adresse : <http://localhost:8080/eVente/eVente-app-client> (chemin à adapter en fonction de votre configuration). Que se passe-t-il ?

Par le même mécanisme, vous devriez pouvoir exécuter l'application déployée sur le serveur de vos condisciples.

2 CONTENEUR D'APPLICATION CLIENTE

Votre application cliente est écrite dans un « conteneur d'application cliente ». Cela permet de faciliter son écriture grâce au mécanisme d'injection : des annotations sont reconnues par le conteneur qui initialise des objets (liés aux sessions beans) pour vous.

Pour cela, l'application ne doit pas être lancée directement mais c'est le conteneur qu'il faut lancer²; il se chargera de lancer et de contrôler votre application. A nouveau Netbeans s'en charge pour vous mais cela peut être fait manuellement via le script « appclient ».

Notez que, pour des raisons qui m'échappent, seule la classe principale peut bénéficier des annotations.

¹Mettre une application au format « Web Start » n'est pas très difficile; il suffit de lui ajouter un fichier XML de description (JNLP)

²Vous avez probablement déjà tous rencontré un « NullPointerException » en tentant d'exécuter directement la classe principale de votre client.

3 DIALOGUE CLIENT-SERVEUR

Si votre application cliente n'est pas gérée par un conteneur il vous faudra gérer vous-même la connexion au serveur. Voyons quelles sont les technologies mises en oeuvre.

Au commencement étaient les ports

Comme vous l'avez probablement vu au cours de système, le dialogue client-serveur se fait à la base via les ports (sockets). Cela nécessite un lourd travail d'encodage/décodage des messages sur base d'un protocole propre à chaque application. Ce sera aussi le cas ici mais ce sera caché par des protocoles de plus haut niveau.

Puis vint RPC (Remote Procedure Call)

RPC est un protocole qui repose sur les ports et qui permet à un client d'appeler une procédure (on est encore en non orienté objet) du serveur. Tout le travail d'encodage/décodage est automatique et caché.

Corba (Common Object Request Broker Architecture)

Pour faire simple, ce protocole est le pendant OO de RPC. Il permet le dialogue entre applications objets hétérogènes (tant au niveau du système que du langage). Ainsi, un client sur un OS donné et écrit dans un langage donné peut utiliser un objet sur un autre système et potentiellement un autre langage.

RMI (Remote Invocation Method)

Une solution 100% Java pour le dialogue client-serveur. Elle permet à un objet Java sur une machine d'appeler une méthode d'un objet Java sur une autre machine. Des passerelles entre Corba et RMI sont développées mais cela n'est pas encore très clair pour moi...

JNDI (Java Naming & Directory Interface)

Avant de pouvoir appeler une méthode sur un objet distant, il faut localiser cet objet et obtenir une référence vers celui-ci. En Java, cela se fait via le protocole JNDI qui peut être vu comme un annuaire. Un serveur JNDI connaît une association entre un nom (par exemple: ejb/GestionFacadeBean) et une « ressource » (un objet, une BD, ...). L'espace de nom est hiérarchisé, c'est ainsi que, par convention, les session beans sont dans « ejb/ ».

4 UN CLIENT HORS CONTENEUR

Voyons à présent ce qu'il faut mettre en oeuvre pour qu'un client accède à des sessions beans.

La première étape est de créer un « contexte JNDI » (en gros d'entrer en contact avec un serveur JNDI)

```
Context ctx = new InitialContext();
```

Les informations sur le serveur (son url par exemple) sont trouvées dans un fichier de configuration (jndi.properties). Par défaut, on contacte le serveur local.

Le session bean s'obtient via une requête « lookup » adressée au serveur JNDI

```
CatalogRemote catalogue = (CatalogRemote) ctx.lookup (« ejb/Catalog »);
```

On est à présent revenu dans la même situation que dans le mode « géré par le conteneur »; on dispose d'un objet « session » dont on peut appeler les méthodes « métiers ».

Exercice

Créez un projet de type « Application Java » qui fait la même chose que votre application cliente développée dans le projet « Entreprise ». Il faudra veiller aux points suivants

- Ajoutez le module ejb dans la librairie
- Ajoutez aussi : j2ee.jar et appserv-rt.jar
- Consultez les fichiers de configuration dans le module ejb pour trouver le nom JNDI donné à vos EJB. Si il n'y en a pas, utilisez le nom complet de la classe
Ex: mcd.ad3.evente.business.GestionUtilisateurRemote

Quand cela fonctionne, portez votre client sur une autre machine et configurez le contexte JNDI pour qu'il accède au serveur.

```
Properties props = new Properties();  
props.setProperty("org.omg.CORBA.ORBInitialHost", "adresse_serveur");  
Context ctx = new InitialContext(props);
```