Découvrez JavaEE 5 avec NetBeans 5.5 - partie 1

par

Date de publication : 17/06/06

Dernière mise à jour : 17/06/06

L'objectif de cet article est d'expliquer certains principes des EJB 3 de Java EE 5 et de les illuster par l'utilisation de NetBeans 5.5 qui permet en quelques minutes de créer un premier Bean, de le déployer et de le tester. Le but n'est pas de fournir un tutorial complet sur Java EE 5 ni sur NetBeans 5.5 mais de fournir quelques bases pour comprendre et utiliser les deux. Ceci est plutôt une mise à plat d'un ensemble de connaissances pour poser les premières pierres de notre ouvrage et illustre la facilité d'utilisation de JavaEE 5. La connaissance préalable de Java EE n'est pas nécessaire mais une bonne connaissance de Java SE 5 est un minimum requis. Durée prévue : 30 minutes.

I - Logiciels nécessaire pour utiliser ce tutorial

Avant de commencer, vous devez installer les logiciels suivant sur votre ordinateur:

- NetBeans IDE 5.5 (télécharger). La version 5.5 Beta contient l'ensemble NetBeans IDE + Sun Java System Application Server 9 et est le meilleur choix pour un débutant car il suffit d'installer et tout est prêt.
- Java Standard Development Kit (JDK) version 5 (télécharger)

II - Notations utilisée dans ce tutorial

<GLASSFISH_HOME> - le répertoire d'installation du serveur d'application Sun Java System Application Server 9 (en général c:\Sun\AppServer) ou du serveur Glassfish

III - Un bean calculatrice avec un EJB 3

Illustrons par un premier exemple concret : nous allons créer un EJB servant de calculatrice pour faire une addition de deux entiers.

III-A - Rappels

Un Entreprise Java Bean 3 (EJB 3) est conforme aux spécifications JavaEE 5. Pour être clair, un Entreprise Java Bean n'a en commun avec les Beans traditionnels que les getters et les setters. Pour le reste, il fonctionne à l'intérieur d'un conteneur EJB qui lui apporte de nombreuses possibilités supplémentaires souvent nécessaire dans le cadre des applications d'entreprise : sécurité, transactions, persistence, réseau, optimisation de l'utilisation de la mémoire, services web, clustering et bien d'autre. La version 3 est une nouvelle version axée sur la simplification et la productivité. Elle tire les leçons du passé et du succés de certains frameworks concurrents. Le conteneur EJB est un élément d'un serveur d'application Java Entreprise Edition 5. Ce dernier comprend aussi un conteneur Web dont le plus connu est Tomcat. Le serveur d'application peut aussi intégrer des accés aux applications traditionnelles par des connecteurs. Les spécifications Java EE 5 permettent de garantir que chaque composant peut fonctionner sans recompilation dans un serveur d'application conforme et laissent toute liberté aux différents fournisseurs sur l'implémentation. Le choix du fournisseur n'est donc pas figé et se fait suivant les qualités supplémentaires que celui-ci est capable d'apporter tout en restant compatible. Le choix d'un bean calculatrice permet de monter que l'on peut faire des choses simple de façon simple, ce qui n'était pas si évident avec les précédentes versions Java EE.

III-B - Création du projet

Aprés avoir installé NetBeans 5.5 et le serveur d'application Sun Java System Application Server 9 nous démarrons NetBeans 5.5. Nous commencons par créer un nouveau projet par le menu File > New Project, nous choisissons Entreprise et EJB Module puis Next. Laissez le nom par défaut (ce sera quelque chose comme EJBModule1), indiquez le répertoire où vous désirez placer votre projet et assurez vous que la version est bien Java EE 5 (si ce n'est pas le cas, aucun serveur Java EE 5 n'est enregistré, il faut alors choisir l'onglet Runtime, puis avec le click droit sur Server choisir Add Server et indiquer le serveur et son répertoire d'installation) et cliquez sur Finish.

III-C - Création d'un Bean Session

Dans l'onglet Projet, ouvrez le noeud EJBModule1 pour voir le noeud Source Package et avec le click droit de la souris sur ce dernier choisissez New puis Session Bean. Dans la zone EJB Name, indiquez Calculatrice et dans la zone package indiquez demo.ejb3.calculatrice. Choisissez un type de Session Stateless et la création d'une interface de type Remote (désélectionnez local si celui ci est coché). Cliquer sur Finish.

NetBeans créé alors deux sources, un pour l'implémentation de notre calculatrice, l'autre pour l'interface de notre calculatrice. Le source de notre Bean se nomme CalculatriceBean et est affiché : nous sommes prêt à saisir notre implémentation. Pour mieux comprendre les relations, ouvrons le source de l'interface CalculatriceRemote. Pour cela, dans l'onglet Projet, ouvrez les noeud EJBModule1, Source Package, puis notre package demo.ejb3.calculatrice.

Dans ce package, vous trouvez le source de l'interface CalculatriceRemote; double cliquez pour l'ouvrir. Pour le moment, l'interface est une enveloppe vide, difficile de faire plus simple, elle n'étend aucune autre interface et ne contient aucune méthode :

```
@Remote
public interface CalculatriceRemote {
```

```
}
```

Vous pouvez remarquer seulement l'annotation @Remote qui trahit que cette interface est une interface EJB éloignée ainsi que l'import correspondant import javax.ejb.Remote. Les connaisseurs EJB 2.0 apprécieront cette simplicité. Cette interface Remote est destinée à communiquer par l'intermédiaire du réseau et interagir avec le monde extérieur. C'est elle que nous manipulerons côté client sans nous inquiéter de l'implémentation que le conteneur va fournir.

Voir la démo

III-D - Ajout d'une méthode métier

Revenons à notre source CalculatriceBean. Nous remarquons qu'il est aussi trés simple avec une simple annotation @Stateless() qui le différencie d'un bean Java SE et qu'il implémente l'interface CalculatriceRemote.

Nous allons fournir un composant qui est chargé de faire une addition. Pour cela nous ajoutons une méthode dite métier (business method) car nous avons la connaissance par notre métier de la façon d'implémenter une addition. Si la méthode devait calculer le montant de votre déclaration d'impôt, seul des spécialistes dans ce métier seraient apte à donnez la bonne formule. Cette méthode ne doit s'occuper que de la partie logique sans intégrer de notion d'affichage ou de persistence en base de données. L'objectif est de séparer chaque couche en se focalisant sur la couche application qui doit être le plus possible indépendante du reste.

NetBeans permet d'ajouter notre méthode en quelques clics : dans le source CalculatriceBean, cliquer avec le bouton droit pour avoir le menu contextuel et choisir EJB Methods puis Add Business Method... Dans le haut de la fenêtre Add Business Method..., nous indiquons le nom additionner dans la zone Name et int dans le type de retour Return Type. Dans la partie basse, dans l'onglet Parameters, nous cliquons sur Add pour ajouter un paramètre x de type int en indiquant int dans la zone Type et x dans la zone name suivi de OK pour confirmer la fenêtre Enter Method Parameter. Nous cliquons à nouveau sur Add pour ajouter un paramètre y de type int en indiquant int dans la zone Type et y dans la zone Name suivi de OK pour confirmer la fenêtre Enter Method Parameter. Nous confirmons notre méthode en cliquant sur OK. Notre source est un peu plus étoffé avec notre méthode :

```
    public int additionner(int x, int y) {
        //TODO implement additionner
        return 0;
    }
```

Nous pouvous jetter un oeil à notre interface CalculatriceRemote : elle contient aussi notre méthode. NetBeans gère les deux sources et permet de les garder synchronisés sous certaines conditions.

Nous allons maintenant terminer notre CalculatriceBean en remplaçant return 0 par return x + y.

Beaucoup d'explications pour une simple addition et ce n'est pas fini, je n'ai pas expliqué ce qu'est un Bean session de type sans état (stateless) !!! Deployons notre bean pour rester dans l'action. Pour cela appuyons simplement sur la touche F6 ou cliquons le menu Run puis Run Main Project. Notre fourmi Ant va compiler le projet, démarrer le serveur d'application et déployer le module EJB. Un peu de patience... et le message BUILD SUCCESSFUL devrait apparaître. Cela veut dire que de nombreuses opérations ont été réalisées, en particulier le conteneur a créé une infrastructure pour permettre à notre bean de profiter des avantages de Java EE : le bean est accessible par le réseau, il peut être retrouvé par l'intermédiaire de JNDI, il est sécurisé, mis en pool, ne nécessite pas de se préocuper de la synchronisation et peut faire partie d'une transaction, tout ceci simplement par deux annotations. Beaucoup de choses qui nécessiteraient des centaines d'heures de travail si l'on devait les écrire soit même ainsi qu'un livre entier pour entrer dans le détail.

Voir la démo

Tout développeur extrême aurait remarqué que le scénario de test unitaire est manquant. Il est temps d'attaquer cela.

III-E - Le test unitaire de notre calculatrice

Rien de plus simple : depuis le source de CalculatriceBean appuyez sur les touches Ctrl+Maj+U ou utilisez le menu Tools puis Create JUnit Test (le source de CalculatriceBean doit être sélectionné pour que cette option fonctionne). Laissez les valeurs par défaut et cliquer sur OK. NetBeans va créer tout le squelette nécessaire à faire un test de notre Bean.

Dans le source de notre test, commençons par vérifier que notre calculatrice est conforme aux attentes. Eliminons la ligne fail("The test case is a prototype."); . Nous cliquons sur CalculatriceBean dans l'explorateur de Projet et lançons le test par les touches Ctrl+F6 ou par le menu Run, Run File... Test "CalculatriceBean.java". Le test confirme que 0+0 = 0.

Rajoutons quelque chose de plus avancé, tentons de vérifier 2+2 = 4. A la fin de la méthode testAdditionner et nous ajoutons :

```
            x = 2;
            y = 2;
            expResult = 4;
            result = instance.additionner(x, y);
            assertEquals(expResult, result);
```

Lançons de nouveau le test qui se passe bien et qui nous confirme que nous avons encore quelques souvenirs des bases des mathématiques.

Les connaisseurs remarqueront que nous avons testé notre bean de la même manière qu'un simple bean classique Java SE appelé POJO (Plain Old Java Object). L'avantage évident est de pouvoir faire des tests sans le conteneur sur toutes les méthodes qui ne font pas appel aux services de celui-ci. Le gain de temps est énorme car si nous modifions notre classe calculatrice, nous pouvons lancer le test sans redéployer le nouveau Bean dans le conteneur, cette dernière opération étant assez longue comme vous avez pu le remarquer. De plus l'exécution est plus longue à l'interieur du conteneur car elle passe par tous les services : localisations du Bean par JNDI, activation ou création d'un Bean dans un pool, vérification des droits d'accés alors que nous n'en avons pas besoin pour le moment. Cette facilité de test était réclamée depuis longtemps par de nombreux développeurs.

Maintenant que notre test se passe bien hors du conteneur, il faut garantir que cela se passe bien à l'intérieur de celui-ci. Ajoutons maintenant le test de notre bean déployé bien au chaud dans le conteneur. C'est le moment d'annoncer la mauvaise nouvelle : notre test unitaire est un simple programme Java SE hors du conteneur et il ne bénéficie pas de tous les avantages. Le conteneur ne va pas automatiquement fournir la référence à notre bean, il faut suivre la voie hiérarchique et passer par le service JNDI qui permet de lui demander un objet en fonction de son nom. Nous verrons que tout ce code est inutile si l'on reste dans le conteneur EJB.

L'appel d'une méthode hors du conteneur nécessite donc plusieurs opérations préliminaires :

- l'obtention du contexte du conteneur pour accéder à ses services : InitialContext ctx = new InitialContext()
- l'obtention d'une référence à un object implémentant l'interface calculatrice capable de transmettre l'appel au conteneur avec les paramètres et de récupérer le résultat, ceci grace à JNDI : Object ref = ctx.lookup("demo.ejb3.calculatrice.CalculatriceRemote");
- le cast de cet objet pour obtenir notre interface, qui doit se faire de façon particulière car nous utilisons une référence éloignée qui passe par un mécanisme de d'appel à distance: CalculatriceRemote calc =

(CalculatriceRemote)PortableRemoteObject.narrow(ref,CalculatriceRemote.class);
- l'appel de notre méthode

Ce code signifie que d'abord nous recherchons l'interface bean et ensuite nous invoquons sa méthode métier. Tout en restant trés proche, les connaisseurs des EJB 2.1 apprécierons de ne pas avoir besoin d'appeler la méthode create et les autres méthodes de gestion du cycle de vie du bean.

L'obtention du contexte du conteneur n'est pas normalisée et est spécifique à chaque conteneur. L'exemple suivant est prévu pour fonctionner avec le serveur Sun Java System Application Server 9 (ou Glassfish) en local. Si le serveur est sur une autre machine, il faut se référer à la documentation développeur de Glassfish.

Dans cet exemple nous utilisons le nom qualifié de l'interface métier qui est le nom JNDI par défault donné par le conteneur.

Commençons par ajouter les biblithèques nécessaires : les interfaces JavaEE et le run time du serveur d'application. Pour cela, cliquez avec le bouton droit sur Test Libraries dans le projet et sélectionnez Add Jar/Folder. Dans le répertoire lib du repertoire d'installation du serveur d'application <GLASSFISH_HOME>, sélectionnez les deux fichiers jar suivant : j2ee.jar et appserv-rt.jar. Ils contiennent tout ce qu'il faut à un client Java SE pour se connecter au serveur.

Ajoutons le code suivant dans notre méthode testAdditionner de CalculatriceBeanTest :

```
          InitialContext ctx;
          try {
              ctx = new InitialContext();
              Object ref = ctx.lookup("demo.ejb3.calculatrice.CalculatriceRemote");
              CalculatriceRemote calc =
(CalculatriceRemote)PortableRemoteObject.narrow(ref,CalculatriceRemote.class);
              expResult = calc.additionner(x,y);
          } catch (NamingException ex) {
              fail(ex.getMessage());
          } catch (RemoteException ex) {
              fail(ex.getMessage());
          }
```

Testez de nouveau le bean, ce coup ci le délai est un peu plus long avant de confimer que le test se passe bien. Si vous cliquez sur le bouton Output de la fenêtre de test unitaire, vous pouvez voir les lignes suivantes qui indiquent que le protocole réseau IIOP est utilisé pour communiquer avec le serveur:

```
9 avr. 2006 18:52:23 com.sun.corba.ee.spi.logging.LogWrapperBase doLog
INFO: "IOP00710299: (INTERNAL) Successfully created IIOP listener on the specified host/port: all
interfaces/1448"
```

Voir la démo

III-F - Qu'est ce qu'un bean session sans état

Le bean que nous avons créé est un bean session sans état. Ce type de bean est le plus simple et le plus efficace. En effet, au lieu de créer un nouveau bean, de l'initialiser puis de le supprimer à chaque appel, le conteneur va mutualiser un pool de beans identiques dans lequel il pioche suivant les besoins et qu'il recycle aprés chaque utilisation. C'est ainsi que le serveur d'application va servir des milliers de requêtes utilisateurs avec seulement quelques objets bean en mémoire. Le cycle de vie d'un objet session bean sans état est trés simple : il passe de l'état de non existence à l'état prêt puis est détruit si le conteneur juge qu'il peut disparaître pour récupérer de la place en mémoire. Si le nombre d'utilisateurs est trés important, le conteneur va créer de nouvelles instances. Lorsque le nombre d'utilisateur diminue et que le conteneur à besoin de mémoire, il va simplement les détruire. Ainsi, vous l'aurez compris, les EJB sont destinés à des applications ayant une valeur ajoutée et prévues pour

évoluer vers des architectures de grande taille.

IV - Conclusion

Nous venons de poser notre première pierre : nous avons obtenu un objet du serveur fonctionnant dans une machine virtuelle différente (et en général sur un serveur différent) sans nous soucier du protocole réseau. L'utilisation d'une interface a rendu tout cela transparent et le conteneur nous épargne toute la programmation des invocations à distance par RMI. Grâce aux EJB 3.0, la programmation de la partie serveur est aussi simple que si l'application devait fonctionner en local et nous pouvons même partiellement tester nos classes sans le conteneur.