Nancy-Université Master IST-IE — 2007–08 — UE 203d

Mise en œuvre de serveurs d'application — TD n° 3

1 Introduction

Dans ce TD, vous implanterez la logique métier de l'application de la bibliothèque imaginaire.

2 Configuration de XDoclet

Dans le premier TD nous avons déjà configuré XDoclet. Néanmoins il reste à indiquer également que xdoclet doit produire les fichiers de configuration pour JBoss.

1. Lancez eclipse (rappel : `/home/prof/burel/eclipse/eclipse`).
2. Allez dans le menu Window d'eclipse, cliquez sur Préférences...
3. Développez XDoclet, cliquez sur ejbdoclet.
4. Cochez la case devant JBoss.
5. Cliquez sur Apply puis sur Ok.

Le support de XDoclet dans eclipse est un petit peu bugué, ce qui va vous obliger à désactiver la compilation automatique. (Normalement, dès qu'un fichier est enregistré, la compilation commence.)

6. Dans le menu Project d'eclipse, décochez Build automatically.
7. À partir de maintenant, pour compiler l'application après avoir sauvegardé un fichier, il vous faudra cliquer sur la quatrième icône sous la barre de menu (celle représentant une feuille avec 010 dessus). Il vous est fortement conseillé de faire ceci deux fois de suite, de façon à empêcher le bug de se produire.

3 Création d'un conteneur d'EJB

Pour écrire la logique métier de votre application, vous allez créer un conteneur d'EJB que vous rajouterez à l'application :

1. Cliquez droit sur TD1_*votre_login*.

2. Choisissez New ▶ EJB Project.
3. Appelez le projet `TD1_`*votre_login*`EJB` .
4. Cochez Add project to EAR. Vérifier que dans EAR Project Name il y a bien le nom de l'application créée au TD n° 1 (normalement TD1_*votre_login*).
5. Cliquez sur Next.
6. Comme vous allez utiliser XDoclet, vous ne pourrez pas utiliser la version 3.0 des modules EJB. Cliquez sur la flèche à droite de Default Configuration for JBoss v4.2. Choisissez EJB Project with XDoclet.
7. Cliquez sur Finish.

Deux nouveaux projets sont créés : TD1_*votre_login*EJB et TD1_*votre_login*EJBClient. L'avantage de séparer le conteneur d'EJB en deux est que les clients de l'application, que ce soit la couche web ou un autre programmme, n'ont besoin de connaître que les interfaces des EJB, qui sont moins grosses à distribuer.

Vous allez indiquez que votre couche web va faire appel aux méthodes des EJBs de votre conteneur.

8. Cliquez droit sur votre projet web (normalement TD1_*votre_login*Web), choisissez Properties.
9. Cliquez sur J2EE Module Dependencies à gauche.
10. Cochez la case de la partie client du conteneur EJB (normalement TD1_*votre_login*EJBClient)
11. Cliquez sur Apply puis sur Ok.

4 Ajout d'un EJB entité

Nous allons créer un Entreprise Java Bean entité qui représentera une donnée livre. Celle ci contiendra les champs suivants :

Champ	cote	titre	retour	emprunteur
Type	String	String	Date	Utilisateur
Remarque	Clé primaire			Clé extérieure

1. Cliquez droit sur TD1_*votre_login*EJB.
2. Choisissez New ▶ XDoclet Enterprise JavaBean.
3. Cochez Container Managed Entity Bean. Cliquez sur Next.
4. Dans Java package, indiquez `biblio` (il est obligatoire de mettre les EJB dans un package autre que celui par defaut).
5. Dans Classe name, indiquez `Livre_`*votre_login*`Bean`. Cliquez sur Next.
6. Dans Usecase. choisissez Define new attributes.

   En effet, nous allons créer une nouvelle table pour le données. De façon plus plausible dans un cas réel où la base de données est déjà remplie, il aurait fallu laisser Import attributes from table.

7. Cliquez sur Next.
8. Dans Table indiquez `Livre_`*votre_login*
9. Cliquez sur Add.
10. Cliquez sur les cases pour changer leur contenu :
    - aField en cote
    - ACOLUMN en COTE
    - 0 (Size) en 8
    - Cochez aussi Primary Key.
11. Cliquez sur Add. Cliquez sur les cases pour changer leur contenu :
    - aField en titre
    - ACOLUMN en TITRE
    - 0 (Size) en 30
12. Cliquez sur Add. Cliquez sur les cases pour changer leur contenu :
    - aField en retour
    - ACOLUMN en RETOUR
    - java.lang.String en java.util.Date
    - VARCHAR (JDBC Type) en DATE (choisir dans la liste)
    - VARCHAR (SQL Type) en DATE (choisir dans la liste)
13. Cliquez sur Finish.
    Le nouveau bean se trouve alors dans le Project Explorer à gauche à la position TD1_*votre_login*EJB ▶ ejbModule ▶ biblio ▶ Livre*votre_login*Bean.java. Double-cliquez dessus pour l'ouvrir.
14. Il vous faut modifier la méthode de création de l'EJB `ejbCreate`. Elle doit prendre en arguments le titre du livre. La cote sera calculée automatiquement et la date de retour sera affectée à `null` pour commencer (ce qui signifie que le livre est disponible). Pour cela :
    (a) recherchez `ejbCreate()` dans le fichier et remplacez le par `ejbCreate(String titre)`.
    (b) entre `// begin-user-code` et `return null`, rajoutez

    ```
    setCote(Livre_votre_loginUtil.generateGUID(this).substring(0, 8));
    setTitre(titre);
    ```

    Comme on peut voir, on utilise la classe d'aide `LivreUtil` générée par XDoclet pour créer un nouvelle cote.
    (c) Sauvegardez et compilez (deux fois).
    Normalement, l'onglet Problems en bas ne devrait contenir que des Warnings et pas d'Errors.
15. Il faut indiquer à JBoss qu'il doit créer la nouvelle table si elle n'existe pas. Recherchez la ligne avec `@jboss.persistence` . Remplacez les trois `false` sur cette ligne par `true`
16. Sauvegardez et compilez (deux fois). Publiez sur le serveur.
    Si vous ouvrez le fichier `/home/prof/burel/nohup.out` (qui contient le journal du serveur JBoss), vous devriez voir apparaître vers la fin une ligne contenant

```
INFO  [EARDeployer] Started J2EE application: file:
/home_PC/burel/jboss-4.2.2.GA/server/default/deploy/TD1_votre_login.ear
```

sans message d'erreur au-dessus.

5 Ajout d'un autre EJB et mise en relation

On va rajouter un autre EJB pour les utilisateurs, et créer la liaison avec les livres. Une donnée utilisateur contiendra les champs suivants :

Champ	nom	adresse	statut	livres
Type	String	String	String	Collection<Livre>
Remarque	Clé primaire			Clé extérieure

1. Cliquez droit sur TD1_*votre_login*EJB.
2. Choisissez New ▶ XDoclet Enterprise JavaBean.
3. Cochez Container Managed Entity Bean. Cliquez sur Next.
4. Dans Java package, indiquez `biblio`
5. Dans Classe name, indiquez `Utilisateur_`*votre_login*Bean. Cliquez sur Next.
6. Dans Usecase. choisissez Define new attributes. Cliquez sur Next.
7. Dans Table indiquez `Utilisateur_`*votre_login*
8. Cliquez sur Add. Cliquez sur les cases pour changer leur contenu :
   - aField en nom
   - ACOLUMN en NOM
   - 0 (Size) en 20
   - Cochez aussi Primary Key.
9. Cliquez sur Add. Cliquez sur les cases pour changer leur contenu :
   - aField en adresse
   - ACOLUMN en ADRESSE
   - 0 (Size) en 20
10. Cliquez sur Add. Cliquez sur les cases pour changer leur contenu :
    - aField en statut
    - ACOLUMN en STATUT
    - 0 (Size) en 10
11. Cliquez sur Finish.

    Le nouveau bean se trouve alors dans le Project Explorer à gauche à la position TD1_*votre_login*EJB ▶ ejbModule ▶ biblio ▶ Utilisateur*votre_login*Bean.java. Double-cliquez dessus pour l'ouvrir.
12. Il vous faut modifier la méthode de création de l'EJB `ejbCreate`. Ici la clé primaire `nom` est passée en argument, ainsi que l'adresse mail et le statut de l'utilisateur. Pour cela :

(a) recherchez `ejbCreate()` dans le fichier et remplacez le par `ejbCreate(String nom, String adresse, String statut)`.

(b) entre `// begin-user-code` et `return null`, rajoutez

```
setNom(nom);
setAdresse(adresse);
setStatut(statut);
```

13. Recherchez la ligne avec `@jboss.persistence` . Remplacez les trois `false` sur cette ligne par `true`
14. Sauvegardez et compilez (deux fois). Publiez sur le serveur. Vérifiez l'absence d'erreur.

Nous allons maintenant créer le lien entre les deux beans entité.

15. Dans Utilisateur_*votre_login*Bean.java, après `public abstract void setStatut(java.lang.String statut);` rajoutez

```
/**
 * @ejb.interface-method viewtype="local"
 * @ejb.relation
 *   name="utilisateur-livres"
 *   role-name="utilisateur emprunte livres"
 */
public abstract java.util.Collection getLivres();

/**
 * @ejb.interface-method viewtype="local"
 */
public abstract void setLivres(java.util.Collection l);
```

16. Dans Livre_*votre_login*Bean.java, après `public abstract void setRetour(java.util.Date retour);` rajoutez

```
/**
 * @ejb.interface-method viewtype="local"
 *
 * @ejb.relation
 *   name="utilisateur-livres"
 *   role-name="livre emprunte par utilisateur"
 * @jboss.relation related-pk-field = "nom"
 *                 fk-column = "emprunteur"
 *                 fk-constraint = "true"
 */
public abstract Utilisateur_votre_loginLocal getEmprunteur();

/**
```

```
     * @ejb.interface-method viewtype="local"
     */
    public abstract void setEmprunteur(Utilisateur_votre_loginLocal l);
```

Pour récuperez les livres empruntés par un utilisateur `u`, il suffit maintenant d'appeler la méthode `u.getLivres()` qui renvoie une `Collection` d'objets de type `Livre_votre_loginLocal`, l'interface locale du bean `Livre_votre_loginBean`.

6 Ajout d'une fonction de recherche sur les livres

Nous allons rajouter la possibilité de rechercher un livre par son titre.

1. Dans Livre_*votre_login*Bean.java, recherchez `@ejb.finder` . Ces trois lignes définit une méthode de recherche sur les livres, nommée `findAll()`, qui les renvoit tous. Par défaut, une autre méthode de recherche, nommée `findByPrimaryKey(String cote)`, permet de trouver le livre dont la cote est `cote`.
2. Nous rajoutons une nouvelle méthode de recherche en fonction du titre en rajoutant le code suivant juste avant `* @ejb.finder ...`

```
*
* @ejb.finder
*  query="SELECT OBJECT(a) FROM Livre_votre_loginSCHEMA a WHERE a.titre LIKE ?1"
*  signature="java.util.Collection findByTitle(java.lang.String titre)"
*
```

   (Il y a donc deux `@ejb.finder` maintenant.)

   Pour rechercher les livres dont le titre contient la sous-chaîne `boutDeTitre`, il suffit maintenant d'appeler la méthode `findByTitle("%" + boutDeTitre + "%")` de l'interface maison de `Livre_votre_loginBean`.
3. Sauvegardez et compilez (deux fois). Publiez sur le serveur. Vérifiez l'absence d'erreur.

7 Ajout d'un bean session

Pour faire un emprunt, il faut connaître des informations à la fois sur le livre (pour vérifier qu'il n'est pas déjà emprunté) et sur l'utilisateur (pour connaître la durée d'emprunt et le nombre de livres autorisés en fonction du statut). Vous allez donc déléguer cette tâche à un bean session qui aura accès au deux.

1. Cliquez droit sur TD1_*votre_login*EJB.
2. Choisissez New ▶ XDoclet Enterprise JavaBean.
3. Cochez Session Bean. Cliquez sur Next.
4. Dans Java package, indiquez `biblio`

5. Dans Classe name, indiquez `Admin_`*votre_login*`Bean`. Cliquez sur Next puis Finish. Le nouveau bean se trouve alors dans le Project Explorer à gauche à la position TD1_*votre_login*EJB ▶ ejbModule ▶ biblio ▶ Admin_*votre_login*Bean.java. Double-cliquez dessus pour l'ouvrir.
6. Tout d'abord, `Admin_votre_loginBean` doit avoir accès au livres et au utilisateurs, grâce aux interfaces locales maison des beans correspondants. Pour cela :
   (a) En début de fichier après `package biblio;` rajoutez `import javax.ejb.*;`
   (b) Après la ligne
   `public abstract class Admin_votre_loginBean implements javax.ejb.SessionBean`
   ajoutez

```
private Livre_votre_loginLocalHome livreHome;
private Utilisateur_votre_loginLocalHome utilHome;
```

   (c) Après la ligne
   `public void ejbCreate() {` ajoutez

```
try {
  livreHome = Livre_votre_loginUtil.getLocalHome();
  utilHome = Utilisateur_votre_loginUtil.getLocalHome();
} catch (Exception e) {}; // ne devrait jamais arriver
```

7. Recherchez `foo(String param)` . C'est la méthode que nous allons modifier pour faire l'emprunt. Remplacez

```
public String foo(String param) {
  return null;
}
```

par

```
public java.util.Date emprunte(String nom, String cote) throws Exception {
  // Recherche de l'utilisateur par son nom
  Utilisateur_votre_loginLocal u;
  try {
    u = utilHome.findByPrimaryKey(nom);
  } catch (FinderException e) {
      throw new Exception("L'utilisateur " + nom
                          + " n'a pas pu être trouvé.");
  };
  // Recherche du livre par sa cote
  Livre_votre_loginLocal l;
  try {
    l = livreHome.findByPrimaryKey(cote);
  } catch (FinderException e) {
      throw new Exception("Le livre de cote " + cote
                          + " n'a pas pu être trouvé.");
```

```
    };
    // Vérification de la disponibilité du livre
    if (l.getRetour() != null) {
       throw new Exception("Le livre de cote " + cote
                           + " est déjà emprunté.");
    };
    // Nouveau calendrier initialisé à la date actuelle
    java.util.Calendar cal = new java.util.GregorianCalendar();
    /* Calcul du nombre de livres maximum à emprunter et de la date
     * de retour en fonction du statut de l'emprunteur
     */
    int mois = 0;
    int maxLivres = 0;
    String statut = u.getStatut();
    if (statut.equals("etudiant")) { mois = 1; maxLivres = 2; }
    else if (statut.equals("doctorant")) { mois = 3; maxLivres = 4; }
    else if (statut.equals("enseignant")) { mois = 6; maxLivres = 8; }
    else {
      throw new Exception("Le statut " + statut + " est inconnu.");
    };
    if (u.getLivres().size() + 1 > maxLivres) {
      throw new Exception("Le nombre maximum de livres empruntés"
                          + " a été atteint.");
    };
    cal.add(java.util.Calendar.MONTH, mois);
    // Emprunt effectif au niveau des beans entité
    l.setRetour(cal.getTime());
    l.setEmprunteur(u);
    return cal.getTime();
  }
```

Comme la base de données que nous utilisons n'existait pas avant, nous allons créer une méthode qui va uniquement servir à la peupler. Nous ajoutons cette méthode dans Admin_*votre_login*Bean.

8. À la suite de la méthode dernièrement écrite, rajoutez le code suivant :

```
  /**
   * @ejb.interface-method
   */
  public void rempliBD() throws Exception {
    Livre_votre_loginLocal l1 = livreHome.create("Les frères Karamazov");
    Livre_votre_loginLocal l2 = livreHome.create("Le père Goriot");
    Livre_votre_loginLocal l3 = livreHome.create("Fondation");
    livreHome.create("1983");
    livreHome.create("Discours de la méthode");
```

```
  utilHome.create("Guillaume", "gb@uhp.fr", "doctorant");
  utilHome.create("Noelle", "nc@uhp.fr", "enseignant");
  utilHome.create("Herve", "hv@n2.fr", "etudiant");
  utilHome.create("Milou", "mi@lou.be", "non renseigné");

  emprunte("Guillaume", l1.getCote());
  emprunte("Guillaume", l2.getCote());
  emprunte("Noelle", l3.getCote());
}
```

8 Lien avec la couche web

Vous allez maintenant faire lien entre la couche web et la logique métier.

1. Dans index.jsp, ajoutez un formulaire pour initialiser la base de donnée à l'aide du code suivant :

```
<h2>Initialisation de la base de données</h2>
<form action="init.jsp">
  <button>Initialiser</button>
</form>
```

2. Créez un nouveau fichier `init.jsp` (cf. TD n° 2). Il contiendra le code suivant :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
...
<%@page import="biblio.*"%>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    ...
  </head>
  <body>
   <% Admin_votre_login a = Admin_votre_loginUtil.getHome().create();
     try {
       a.rempliBD(); %>
    <p> La base de données est initialisée. </p>
    <% } catch (Exception e) { %>
    <p> L'erreur suivante est intervenue :
        <%=e.getMessage()%> </p>
    <% }; %>
  </body>
</html>
```

3. Sauvegardez, compilez (2 fois), publiez, ouvrez firefox et vérifier que le bouton Initialiser fonctionne.

Il faut maintenant modifier emprunts.jsp pour utiliser la logique métier.

4. Ajoutez

```
<%@page import="biblio.*"%>
<%@page import="java.util.*"%>
```

   vers le début du fichier.

5. Remplacez

```
<% // Vecteur de livres créés a la main
  Vector livres = new Vector ();
  livres.add("Les Frères Karamazov");
  livres.add("Fondation");
  livres.add("Du côté de chez Swann"); %>
```

   par

```
<% try {
  Utilisateur_votre_loginLocalHome utilHome =
    Utilisateur_votre_loginUtil.getLocalHome();
  Utilisateur_votre_loginLocal u = utilHome.findByPrimaryKey(util);
  Collection livres = u.getLivres(); %>
```

6. Rajoutez avant `<%@ include file="piedDePage.jspf" %>`

```
  <% } catch (Exception e) { %>
    <p> L'erreur suivante est intervenue :
        <%=e.getMessage()%> </p>
  <% }; %>
```

7. Remplacez `Object livre = iterator.next(); %>` par
   `Livre_votre_loginLocal livre = (Livre_votre_loginLocal)iterator.next(); %>`

8. Remplacez

```
<tr>
  <td> <%=livre %> </td>
  <td> Inconnue </td>
  <td> <a href="retour.jsp?livre=<%=livre%>"> Retour </a> </td>
</tr>
```

   par

```
<tr>
  <td> <%=livre.getTitre()%> </td>
  <td> <%=livre.getRetour()%> </td>
 <td> <a href="retour.jsp?cote=<%=livre.getCote()%>"> Retour </a> </td>
</tr>
```

Ensuite, il faut modifier retour.jsp :

9. Ajoutez

```
<%@page import="biblio.*"%>
<%@page import="java.util.*"%>
```

   vers le début du fichier.

10. Insérez le code suivant entres les deux `<%@ include ...%>` :

```
<% String cote = request.getParameter("cote");
   try {
     Livre_votre_loginLocalHome livreHome =
       Livre_votre_loginUtil.getLocalHome();
     Livre_votre_loginLocal l = livreHome.findByPrimaryKey(cote);
     l.setRetour(null);
     l.setEmprunteur(null);
%>
<p> Le livre <%=l.getTitre()%> a bien été rendu. </p>
<% } catch (Exception e) { %>
<p> L'erreur suivante est intervenue :
    <%=e.getMessage()%> </p>
<% }; %>
```

S'il vous reste du temps, modifiez recherche.jsp pour faire appel à la méthode `findByTitle(titre)` de `Livre_votre_loginLocalHome`, et emprunter.jsp pour faire appel à la méthode `emprunte(nom,cote)` de `Admin_votre_login` (cf fin du TD n° 2).