Fil rouge : séances 3 et 4
Module FIP INF 211

Gestion des services métier et de la couche présentation

Déroulement

Les couches les plus basses de l'application *cabinet de recrutement* sont maintenant en place. Elles sont composées de la base de données ainsi que de la couche données qui y accède. Les fonctionnalités métier accompagnées de la couche qui gèrent l'interface graphique de l'application seront le centre de ces deux séances. Le principe est ici de développer conjointement les éléments de ces deux couches autour d'une fonctionnalité particulière puis de réaliser sur le même modèle le travail pour les autres fonctionnalités.

Chacune de ces deux couches héberge des composants particuliers :

- La couche service qui assure les fonctionnalités métier est constituée d'un ensemble de ***beans session*** (comme le sont d'ailleurs aussi les composants DAO).
- La couche interface graphique est constituée de ***pages JSP*** dont le but est la génération dynamique[1] de contenu Web. Cette couche pourrait aussi être constituée de Servlets Java (comme le *TP architectures*) dont le but est identique mais avec une approche différente.

Une ébauche de ces deux couches existe déjà qu'il faudra compléter dans un premier temps. La suite du travail consistera à compléter l'ensemble des fonctionnalités telles que définies au cahier des charges.

Un document annexe est présent sur Moodle : annexes séances 3 et 4. Il présente la liste des composants développés dans l'application exemple (http://srv-labs-001.enst-bretagne.fr/CabinetRecrutement_WEB/). Les informations qui y sont présentées ne sont fournies qu'à titre indicatif mais elles peuvent constituer une aide à la réalisation de cette étape.

Pour aller plus loin…

- Sur Moodle dans la partie consacrée au module FIP INF 211, partie 4 (*architecture des applications d'un SI*) sont présentées un ensemble de références bibliographiques traitant des technologies JEE : http://formations.telecom-bretagne.eu/fad/course/view.php?id=23428
- Memento HTML : http://nephi.unice.fr/CoursHTML/index.php
- Cours rapide (~30 transparents) sur la technologie JSP : http://www.lifl.fr/~seinturi/middleware/jsp.pdf. Ce cours est assez ancien (2002) mais les principes de base de la technologie sont inchangés.

[1] *Le contenu Web dynamique s'oppose au contenu Web statique : un page Web écrite en HTML a un contenu fixe qui n'évolue pas au contraire d'une page Web dynamique dont le contenu peut évoluer au cours du temps. C'est le cas ici : la liste des entreprises dépend du contenu de la base au moment de la consultation.*

Poursuite du développement du composant *ServiceEntreprise*

La version actuelle de ce composant ne réalise que deux fonctionnalités basiques (qui en fait ne font qu'encapsuler deux fonctionnalités CRUD du DAO *EntrepriseDAO*) :

- ***getEntreprise(int id)*** : obtention d'une entreprise par son id.
  Cette méthode réalise la fonction *afficher les informations d'une entreprise* décrite dans le cahier des charges page 4.
- ***listeDesEntreprises()*** : obtention de la liste de toutes les entreprises référencées dans la base.
  Cette méthode réalise la fonction lister les entreprises référencées décrite dans le cahier des charge page 3.

Examen de l'état actuel de la classe

Le code de la classe se situe dans le projet *CabinetRecrutement_EJB*, au sein du package *eu.telecom_bretagne.cabinet_recrutement.service*. La convention de nommage utilisée dans le fil rouge montre bien que cette classe fait partie de la couche service.

La classe *ServiceEntreprise* est un *bean session*. Comme dans le mapping objet/relationnel en JPA, ceci est paramétré à l'aide d'annotations Java. Cette classe en possède trois :

- ***@Stateless***
  C'est l'annotation principale qui définit la classe comme étant un bean session. C'est un bean sans état, c'est-à-dire qu'il ne mémorise aucune donnée entre deux requêtes.
- ***@LocalBean***
  Cette annotation précise que le bean peut être invoqué directement par sa classe et non uniquement par ces, éventuelles, interfaces (interface locale ou interface distante dite aussi *remote*).
  Note : cette manière de faire est celle qui a été utilisée pour tester le bon fonctionnement des DAO. Ceux-ci ne possédant pas d'interfaces, ils étaient récupérés dans l'interface graphique (via la classe *ServicesLocator*) directement par le nom de la classe. Comme les beans session de la couche service seront récupérés ici à travers leur interface remote, cette annotation est ici optionnelle. Elle a été ajoutée automatiquement par l'assistant d'Eclipse de création des beans session et n'a pas été supprimée ensuite.
- ***@WebService***
  Cette annotation est présente dans le code source mais aurait dû être supprimée. Elle n'est que pour tester la fonctionnalité. Elle permet de transformer le bean en un Web Service sans ajouter une seule ligne de code. Comme cette annotation est présente, il vous est possible de tester le Web Service grâce au testeur de GlassFish à l'URL : http://localhost:8080/ServiceEntrepriseService/ServiceEntreprise?Tester

Le DAO *EntrepriseDAO* est utilisé dans les deux méthodes existantes. Pour le moment, une seule dépendance est nécessaire, ce ne sera plus le cas quand les fonctionnalités seront étendues (voir document annexe). Chaque dépendance est représentée dans la classe de la manière suivante :

- *@EJB private EntrepriseDAO entrepriseDAO;*
  L'annotation *@EJB* précise que la référence vers le DAO sera renseignée par le conteneur au moment du déploiement de l'application.
  D'autre part, le choix opéré ici est un choix plus simple que dans le TP architectures : comme chaque DAO est annoté par *@LocalBean*, il n'est pas nécessaire de typer la référence du DAO (*entrepriseDAO*) par son interface (qui n'existe pas…) mais directement par la classe. Sans l'annotation *@LocalBean*, cela aurait été impossible.

Par choix, il a été décidé que le bean *ServiceEntreprise* puisse être accédé de façon distante pour de futures évolutions (si vous voulez tester ☺…). Ceci n'est pas obligatoire dans notre configuration de déploiement, l'interface Web comme la couche service sont hébergées au sein du même GlassFish. Les méthodes du bean sont donc présentes au sein de l'interface (dans le sens langage Java) distante dont le code source se trouve dans le projet *CabinetRecrutement_EJBClient*.

L'interface et la classe concrète partagent le même package : *eu.telecom_bretagne.cabinet_recrutement.service*. Le nom de l'interface est : *IServiceEntreprise*.

Ajout de nouvelles fonctionnalités

L'ajout de nouvelles fonctionnalités dans le composant *ServiceEntreprise* se traduit matériellement par l'ajout de nouvelles méthodes dans le bean. Comme le bean est accédé via son interface distante, il est nécessaire d'ajouter d'abord la signature de la méthode au sein de l'interface. Une erreur apparaît alors dans la classe concrète : il faut y ajouter l'implémentation de la nouvelle méthode.

Ajout de nouveaux composants

Création d'un bean session

Ajouter de nouveaux composants revient à ajouter de nouveaux beans session. Eclipse dispose d'un assistant qui permet d'automatiser cette tâche. Pour y accéder :

- Sélectionnez le projet *CabinetRecrutement_EJB* dans l'onglet *Project Explorer* ou *Navigator*, clic droit > *New* > *Session Bean (EJB 3.x)*
  Un assistant composé de deux pages apparaît (en cas de doute, il est possible de revenir en arrière).
- **Page 1**
  Plusieurs éléments à renseigner :
  - Le nom du package : *eu.telecom_bretagne.cabinet_recrutement.service*
  - Le nom de la classe : la convention utilisée (à suivre, ou pas…) jusqu'à présent fait précéder le nom du bean par *Service*.
  - *State type:* laissez à *Stateless*.
  - Partie *Create business interface* : cocher *Remote*, un nom pour l'interface est proposé du type *NomDuBeanRemote*. Toujours par convention, il est souvent préféré de préfixer une interface par le caractère 'I' qui rend compte du caractère fonctionnel (I pour interface). Comme il n'est pas prévu d'avoir une interface locale (les interfaces des beans sont toutes distantes), le suffixe *Remote* peut être oublié. Le nom de l'interface devient alors : *INomDuBean*.
  - Laissez cochée la case *No-interface* qui permet l'ajout de l'annotation *@LocalBean*.
- **Page 2**
  Rien à modifier ici, bouton *Finish*.

Vérifiez que la classe et l'interface remote soient bien créées dans le bon package et les bons projets.

Référencement du nouveau bean dans la classe ServicesLocator

Une fois le bean créé, avant de le tester (au sein de la Servlet *ControlesDAOServlet*) ou de l'utiliser dans l'interface Web de l'application, il est nécessaire de le référencer dans la classe *ServicesLocator*. Le commentaire de la méthode *getRemoteInterface* donne des indications pour composer le nom JNDI nécessaire pour retrouver le bean. À noter, les noms JNDI des beans publiés dans le serveur GlassFish apparaissent dans le fichier de log du serveur.

La couche présentation : l'interface graphique de l'application

L'ébauche de l'interface graphique de l'application utilise la technologie JSP (*Java Server Pages*) que l'on peut comparer aux technologies PHP ou ASP (Microsoft). Dans toutes ces technologies le code source de ces pages mêle, d'un point de vue syntaxique, à la fois du code de présentation Web (*HTML* principalement mais aussi éventuellement du *code CSS* et/ou *Javascript* et/ou etc.) et du code Java notamment pour les appels aux beans session précédemment développés.

La durée du module ne permet pas d'assurer un cours sur les interfaces Web mais, au moins à un premier niveau, la technologie est relativement simple. Cette partie vise à expliquer l'existant déjà fourni à partir duquel il sera possible d'écrire les interfaces pour les nouvelles fonctionnalités.

L'application du TP architectures n'utilise pas de pages JSP mais des Servlets Java. Si vous le préférez, il est aussi tout à fait possible d'utiliser cette approche. Dans le cas de la technologie JSP, ce qui est réellement exécuté côté serveur est une Servlet. En effet le serveur génère à partir du code de la JSP le code source de la Servlet équivalente (c'est une sorte de compilation), cette Servlet est ensuite compilée (génération d'un fichier *.class*) puis exécutée. Ce processus est exécuté lors de la première invocation de la page JSP. Il est peut être utile de consulter ce code source, la pile d'erreurs (*stacktrace*)peut y faire référence. Le code Java des Servlets générées se trouve dans le répertoire */opt/glassfish3.1.1/glassfish/domains/domain1/generated/jsp/CabinetRecrutement_EAR/CabinetRecrutement_WEB_war/org/apache/jsp*

Rappel : le fil rouge n'est pas un concours de design Web, l'essentiel est de pouvoir avoir accès aux fonctionnalités de l'application même si votre interface est un peu « rustique ».

Examen de l'existant

Le codage de l'interface graphique de l'application est l'objet du projet *CabinetRecrutement_WEB*. On y trouve deux parties principales :

- Le contenu Web en lui-même à l'intérieur du répertoire *WebContent* : pages JSP, pages HTML mais aussi fichiers CSS, fichiers images, etc. Tout ce qui est déposé ici est accessible via le serveur à l'URL http://localhost:8080/CabinetRecrutement_WEB/. C'est le répertoire de base du contenu Web.
  À noter : tout fichier possédant l'extension *.jsp* est considéré comme une page JSP et sera donc compilé lors de sa première invocation, compilation qui peut éventuellement échouer.
- Le code Java d'un certain nombre de classes utilisables par les pages JSP (répertoire *src*). Ces classes appartiennent au sous package *eu.telecom_bretagne.cabinet_recrutement.front* :
  - Le code de la Servlet permettant de tester les fonctionnalités des composants EJB : *ControlesDAOServlet*.
  - La classe utilitaire qui permet de retrouver les références des composants EJB de la couche métier : classe *ServicesLocator*.
  - Un classe utilitaire (*Utils*) qui fournit des fonctionnalités pour la manipulation des dates (format *java.util.Date* <--> *String* : méthodes *string2Date* et *date2String*) et pour la gestion des retours chariot en HTML (elle remplace les '\n' en leur équivalent HTML : '*
*').

Trois JSP existent à ce stade du projet :

- *index.jsp* : page d'affichage du menu.
- *liste_entreprises.jsp* : affiche la liste de toutes les entreprises référencées dans la base.
- *infos_entreprise.jsp* : affiche le détail des informations d'une entreprise donnée.

Focus sur le JSP liste_entreprise.jsp

L'examen de ce code source permettra de comprendre le principe de base du codage d'une page JSP qui pourra être reproduit.

- Ligne 1 : on précise que cette JSP est écrite en Java, et que l'encodage des caractères correspond aux langues de l'Europe de l'ouest (*ISO-8859-1*). Cette ligne est à copier/coller lors de la création d'une nouvelle JSP.
- Lignes 3 à 6 : une page JSP contient du code Java. Certaines parties de ce code nécessitent une visibilité et, comme toutes les classes (ce JSP en deviendra une), il est nécessaire de les importer. Toutes les importations sont séparées par une virgule.
- Lignes 8 à 13 : on trouve du code Java. L'écriture de code Java dans une JSP est encadrée par *<% … %>* (en PHP l'équivalent est *<?php … ?>*). C'est du code java « classique » : le JSP récupère la référence vers le composant EJB *ServiceEntreprise* (à l'aide de *ServicesLocator*) puis s'en sert pour obtenir la liste des entreprises par l'appel de la méthode *listeDesEntreprises()*. Cette liste est référencée dans la variable *serviceEntreprise* qui a une visibilité sur l'ensemble du JSP.
  Le code Java encadré par *<% …code Java… %>* est appelé un *scriplet*.
- Lignes 15 à 32 : début du codage HTML (basique…) de l'interface. Cette partie est le codage statique de la page. À noter :
  - Ligne 20, le lien vers le fichier CSS qui code la mise en forme de l'interface (couleur de fond, polices de caractères utilisées, mise en forme des tableaux, etc.). Ce fichier CSS vous est fourni, il vous est possible de l'utiliser et éventuellement de le modifier pour avoir une charte graphique cohérente sur l'ensemble de l'application.
  - Ligne 27 : le tableau affichant la liste des entreprises fait référence à une mise en forme particulière définie dans le fichier CSS (tableau de type *affichage*).
  - Ligne 27 : le codage HTML d'un tableau débute par *<table>* et se termine par *</table>*. Une ligne du tableau est encadrée par *<tr>* et *</tr>* (pour la ligne d'entêtes des colonnes : *<th>* … *</th>*). Une cellule (d'une ligne d'un tableau) est encadrée par *<td>* et *</td>*.
- Ligne 33 : début du code Java qui affiche dynamiquement le contenu de la liste des entreprises. Cette liste, récupérée ligne 12, est parcourue à l'aide d'une boucle. Pour chaque entreprise de la liste, une ligne du tableau HTML est créée (*<tr> …contenu… </tr>*).
  L'affichage de chaque information atomique (nom, adresse, etc.) de l'entreprise est réalisé dans une cellule de tableau à l'aide d'une déclaration JSP de la forme *<%= …contenu à afficher… %>*. Voir, par exemple, l'affichage de l'adresse de l'entreprise ligne 40.
- Lignes 45 à 51 : fin du codage HTML statique de la page.

Gestion des formulaires

La saisie des données via l'interface se réalise à l'aide de formulaires HTML. Le support référencé en début de ce document aborde cette technique (transparent 146 et 147).

Divers

Le transparent 145 présente un certain nombre d'objets dits *objets implicites*. Ces objets sont des instances pré-déclarées qu'il n'est pas nécessaire d'instancier (elles l'ont déjà été par le conteneur Web). Parmi ces objets implicites, vous aurez éventuellement besoin de :

- Objet *out* : permet d'écrire dans la page. Par exemple : *<% out.println(entreprise.getId()); %>* (ce qui est complètement équivalent à *<%=entreprise.getId()%>*.

- Objet *request* : il représente la requête envoyée au serveur. Son utilisation principale est la récupération des paramètres (envoyés par formulaire ou par URL). Par exemple, la liste des entreprises (JSP *liste_entreprises.jsp* ligne 39) génère, par entreprise, un lien HTML pour l'affichage des informations détaillées :
  http://localhost:8080/CabinetRecrutement_WEB/infos_entreprise.jsp?id=1
  Dans l'URL, le paramètre est situé après le caractère '*?*' et se nomme *id*.
  La récupération de la valeur (ici *1*) se fait dans le JSP infos_entreprise.jsp à la ligne 11 grâce à la méthode *getParameter("nom_d_parametre")* de l'objet *request*.
- Objet *session* : cet objet permet la gestion des sessions pour un utilisateur. Grâce à cet objet, il est possible de mémoriser des informations, pour un utilisateur donné, entre deux requêtes (deux pages JSP, etc.). Les données stockées peuvent être de toute nature suivant la nature de l'application Web : caddy virtuel pour un site de vente en ligne, bon de commande, etc. Dans notre cas, il sera possible de stocker dans la session l'utilisateur qui se connecte (soit une entreprise soit un candidat). Trois méthodes principales :
  - *session.setAttribute("nom_de_l_objet_dans_la_session", valeur)*
    Dépôt de la donnée. La valeur est stockée sous la forme d'une instance d'*Object*.
  - *Object contenu = session.getAttribute("nom_de_l_objet_dans_la_session")*
    L'objet possède le type le plus générique (*Object*). Il faut le transtyper (caster) pour pouvoir l'utiliser. Dans notre cas, stockage de l'utilisateur connecté, si l'objet vaut *null*, c'est qu'il n'y a aucun utilisateur connecté. Il est alors facile de réaliser des tests.
  - *session.invalidate()*
    Réinitialisation de la session qui pourra être utilisée dans notre cas pour la déconnection.

  Note : un exemple de gestion des sessions dans le corrigé vous est donné sur Moodle : la page JSP gérant les connexion au système (page *connexion.jsp*), la deconnexion (page *deconnexion.jsp*) et un exemple d'utilisation (page affichant l'utilisateur connecté : *bandeau.jsp*).
  La page JSP *connexion.jsp* constitue aussi un exemple de gestion de formulaire (formulaire simple de saisie de l'identifiant).