TP2 JDBC et JSP
25/03/2010 C.Plumejeaud

1. Configurer un logger

Dans une servlet, **System.out.println** sort dans le fichier **catalina.out** du répertoire de log de votre container. Si toutes les webapps font des traces de cette manière, ce fichier peut devenir illisible.
Avec log4j, il est possible de paramétrer une sortie des logs dans un autre fichier : il suffit ensuite d'utiliser _log.<level> pour récupérer une trace lisible, dans ce fichier.
Toutes les infos sur log4j sont en ligne :
http://logging.apache.org/log4j/1.2/manual.html
http://beuss.developpez.com/tutoriels/java/jakarta/log4j/

La configuration des logs se fait avec le fichier **servletLog.properties**, qui doit être déployé dans la servlet :

```
# Set the level of the logger named "TP[n]" and its sub hierarchy to
# Level.DEBUG, attach appender A2.

log4j.logger.TP1=DEBUG, A2
log4j.logger.TP2=DEBUG, A2

# Appender A2 writes to the file "logs" in user's home.
log4j.appender.A2=org.apache.log4j.FileAppender
log4j.appender.A2.File=${user.home}/webapp-logs.txt

# Truncate 'test' if it aleady exists.
log4j.appender.A2.Append=false

# Appender A2 uses the PatternLayout.
log4j.appender.A2.layout=org.apache.log4j.PatternLayout
log4j.appender.A2.layout.ConversionPattern=%d{dd/MM/yyyy HH:mm:ss} %-5p [%c{2}] %m%n
```

1.1 Les niveaux de log

Il existe plusieurs niveaux de log : du plus bas au plus haut, ce sont :
DEBUG, INFO, WARN, ERROR, FATAL.
Plus on veut de traces, plus il faut descendre le niveau, et les traces de niveau debug contiennent ainsi aussi les traces de niveau supérieur (info, warn, etc.)
Le niveau de trace de chaque paquetage et meme chaque classe peut être controlé sur le ligne de syntaxe suivante :
`Log4j.logger.<package>.<classname> = <level>, <appender>`
Par exemple, `log4j.logger.TP1=DEBUG, A2`
Signifie que toutes classes du package TP1 vont loguer au niveau DEBUG dans la sortie A2

1.2 La sortie du log

On peut configurer plusieurs sorties pour les logs. Ce sont les *appenders*.
Ici A2 est une sortie de type un fichier : `log4j.appender.A2=org.apache.log4j.FileAppender`
Pour spécifier les chemin vers le fichier de log, on utilise la commande suivante :
`Log4j.appender.<nom>.File = path`
Par exemple, `log4j.appender.A2.File=${user.home}/webapp-logs.txt`
Signifie que la sortie A2 de type fichier se fait dans le fichier webapp-logs.txt dans le HOME directory de l'utilisateur (Attention, celui qui a lancé le container !!)
On peut spécifier aussi le format de sortie, en utilisant des règles (`log4j.appender.A2.layout.ConversionPattern`).

1.3 Utiliser le log dans une servlet

Pour utiliser un logger dans une servlet:

1. Déclarer _log comme variable de classe

```
private static Logger _log = null;
```

2. Instancier le logger dans la méthode init() de la servlet

```
//get the path to the loggerPropertie file inside the servlet
String loggerProperties = this.getServletContext().getInitParameter("loggerProperties");
loggerProperties = getServletContext().getRealPath("/") + loggerProperties;
//load the configuration for this application's loggers using the
// servletLog.properties file
PropertyConfigurator.configure(loggerProperties);
//create the logger for this servlet class
//it will use the configuration for the logger com.java2s.LoggerServlet
//or inherit from the logger com.java2s if one exists, and so on
_log = Logger.getLogger(this.getClass());


_log.info(this.getClass().getName()+" started.");
//display a DEBUG level message
_log.debug("Sending a DEBUG message");
//display an INFO level message
_log.info("Sending an INFO message");
//display an ERROR level message
_log.error("Sending an ERROR message");
```

- Editez le fichier **servletLog.properties** et changez le niveau de log de la classe **JDBCServlet**
- Trouvez le fichier de log et vérifiez le format de sortie.

2. Ma première connexion JDBC

Normalement, depuis le TP1, vous avez une base **insee_cog** sur postgreSQL. Sinon, installez le backup disponible à l'adresse suivante : http://stid-grenoble.xtek.fr/index.php?dossier_nav=1000&action=lire&id=2778

2.1 Connecter sur une base de données

La servlet **JDBCServlet** doit lister les entrées de la table **régions**. Elle récupère les paramètres de connexion dans le fichier web.xml, et les analyse au moment de l'initialisation de la servlet :

```
driver = this.getServletContext().getInitParameter("driver");
database = this.getInitParameter("database");
connectString = this.getServletContext().getInitParameter("DBURL")+database;
login = this.getServletContext().getInitParameter("login");
password = this.getServletContext().getInitParameter("password");
```

_ Adaptez les paramètres de connection de JDBCServlet dans le fichier web.xml pour pouvoir vous connecter sur votre base de données **insee_cog**
_ Vous noterez qu'il vous faut le **driver** : il est à télécharger sur l'adresse suivante et à installer dans WEB-INF/lib : http://jdbc.postgresql.org/download/postgresql-8.4-701.jdbc4.jar

2.2 Interroger une base de données

Une autre base de données, **Servlet**, existe : il suffit de l'importer dans postgreSQL, avec le fichier backup disponible sur l'URL suivante http://stid-grenoble.xtek.fr/index.php?dossier_nav=1000&action=lire&id=2779.

Créer une seconde Servlet, **JDBCServletOptimized** à partir du modèle de la première et adaptez la pour exécuter la requête suivante sur la base de données « Servlet » :

```
SELECT * from gens
```

Et faites afficher le contenu sous la forme d'un tableau simple :

login	password	Est admin ?

2.3 Optimiser l'utilisation de la connexion

L'objet « `Connection dbConnection` » est *thread safe* : il peut donc être utilisé par plusieurs accès simultanés sur la Servlet, et être placé dans le contexte mémoire de la Servlet, pas seulement de celui d'une requête. Ce qui veut dire que l'ouverture et la clôture de la connexion peuvent être réalisées en dehors de son utilisation en réponse à une requête. Ceci économise du temps d'accès sur la base de données, car l'établissement d'une connexion est long.

Dans `init() de` **JDBCServletOptimized**, rajoutez du code qui ouvre la connection, et un log qui indique que cela a bien marché.
Dans `destroy() de` **JDBCServletOptimized**, fermez la connection et faites un log qui indique que cela a bien marché.

3. Transformer les pages statiques en pages dynamiques

3.1. La servlet login

La servlet **LoginServlet** qui est dans votre projet vérifie que le login et le password passés en paramètres de la requête correspondent à un utilisateur de la table **gens** dans la base de données **Servlet**. La page login.xhtml propose le formulaire d'authentification (cf. Figure 1) http://localhost:1977/TPServlet/login.xhtml

Figure 1. Formulaire d'authentification

Si l'authentification réussie, elle renvoie l'utilisateur sur la page d'Accueil (Accueil.xhtml), cf Figure 2. Elle renvoie sinon au formulaire d'authentification, avec un message d'erreur.

Figure 2. Page d'accueil

_ faites passer des paramètres qui sont corrects : vous pouvez tester avec
http://localhost:1977/TPServlet/authenticate?login=Daniel&password=wxcvbn78
_ si l'authentification réussit, créer un bean **User** de session
_ utilisez ce bean pour remplir la page accueil.xhtml : transformez accueil.xhtml en page **accueil.jsp**, afin que la phrase suivante soit complétée par les informations venues de l'authentification : Bonjour <login>, vous êtes <etudiant>|<admin>
_ Le lien « Gestion des utilisateurs » ne doit apparaître que si l'utilisateur connecté est admin.
_ Le formulaire de changement de password renvoie vers l'URL relative à la webapp /changePassword : compléter la Servlet **ChangePasswordServlet*** pour qu'elle vérifie le formulaire, et mette à jour le mot de passe dans la base de données. Elle renvoie toujours sur accueil.jsp, mais en cas d'échec, un message d'erreur est affiché (vous utilisez vous cela l'inclusion de la page **message.jsp**)

* Dans le squelette qui vous est fourni, la servlet fonctionne avec un groupe (*pool*) de connexion, qui sont utilisées en mode transactionnel. La documentation suivante explique comment l'utiliser (http://jdbc.postgresql.org/documentation/84/ds-ds.html)

Le bean User :

```
public class User {

        private String login;
        private String password;
        private boolean admin;

        public String getLogin() {
           return login;
        }
        public void setLogin(String login) {
           this.login = login;
        }

        public String getPassword() {
            return password;
        }
        public void setPassword(String password) {
            this.password = password;
        }

        public boolean isAdmin() {
            return admin;
        }
        public void setAdmin(boolean admin) {
            this.admin = admin;
        }

}
```

3.2 Une application complète

La page de gestion des utilisateurs est pour l'instant un formulaire html qui soumet ses paramètres à l'URL /manageUser qui est mappée avec la servlet **ManageUserServlet**. Un paramètre "**action**" permet de déterminer quelle est l'action à entreprendre.
Depuis cette page, l'utilisateur peut faire les actions suivantes :

- Trouver un utilisateur par son login (action=search)
- Lister tous les utilisateurs (action=list)
- Supprimer un utilisateur (action=delete)
- Créer un nouvel utilisateur (action=add)

Le formulaire (cf. figure 3) est pour l'instant statique.
À chaque fois que l'action réussit, l'utilisateur revient sur la page de gestion mise à jour. En cas d'échec, il revient sur cette page, mais un message d'erreur lui est affiché.

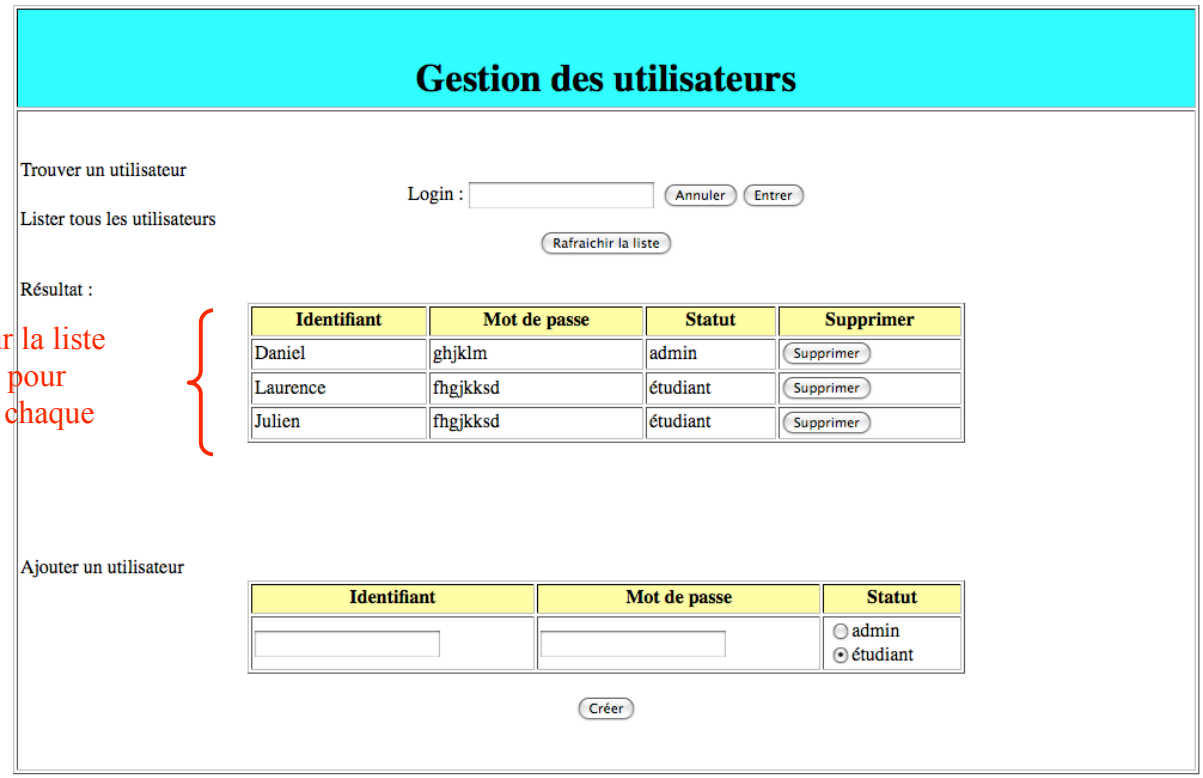

Figure 3. Formulaire de gestion des utilisateurs

_ Il faut transformer la page gestion.xhtml en page dynamique **gestion.jsp**. Elle doit consulter un bean (la liste des utilisateurs en base (*userList*)), et passer en paramètre le login, (password et statut le cas échéant) de l'utilisateur à trouver, créer, supprimer (user).
_ La première fois, la liste des utilisateurs est vide. Il faut cliquer sur « Rafraîchir la liste » pour voir tous les utilisateurs.
_ Vous devez créer la servlet **ManageUserServlet** (en utilisant le modèle ChangePasswordServlet) et **développer toutes les actions**. Mais toutes les actions de consultation et mise à jour de la base de données doivent implémenter l'interface **UserDAO**.
_ Utilisez la page message.jsp pour faire afficher des messages d'erreur !

NOTE : il est conseillé de travailler de façon itérative et de réaliser en premier l'action « find » qui affiche un utilisateur trouvé dans le tableau.

L'interface UserDAO :

```
public interface UserDAO {

          public boolean insertUser(User u);
          public boolean deleteUser(User u);
          public ArrayList<User> findUser(String login);
          public boolean updateUser(User u);
          public ArrayList<User> allUsers();

}
```

Au final, lorsque vous aurez fini le TP, vous aurez une webapp complète qui permet de gérer des utilisateurs et sur le mode Modèle/Vue/Contrôleur.
Le Modèle est porté par le bean User : il représente les données manipulées.
La vue, ce sont les pages de présentation des informations : Login.html, accueil.jsp, gestion.jsp et modify.jsp
Le Controller, ce sont les 3 servlets : LoginServlet, ChangePasswordServlet, et ManageUserServlet. Le diagramme d'activité de la figure 3 présente l'enchaînement des activités (rectangle aux bords arrondis) et des pages de l'application (rectangle aux bords carrés). Les paramètres qui transitent apparaissent dans les notes informatives

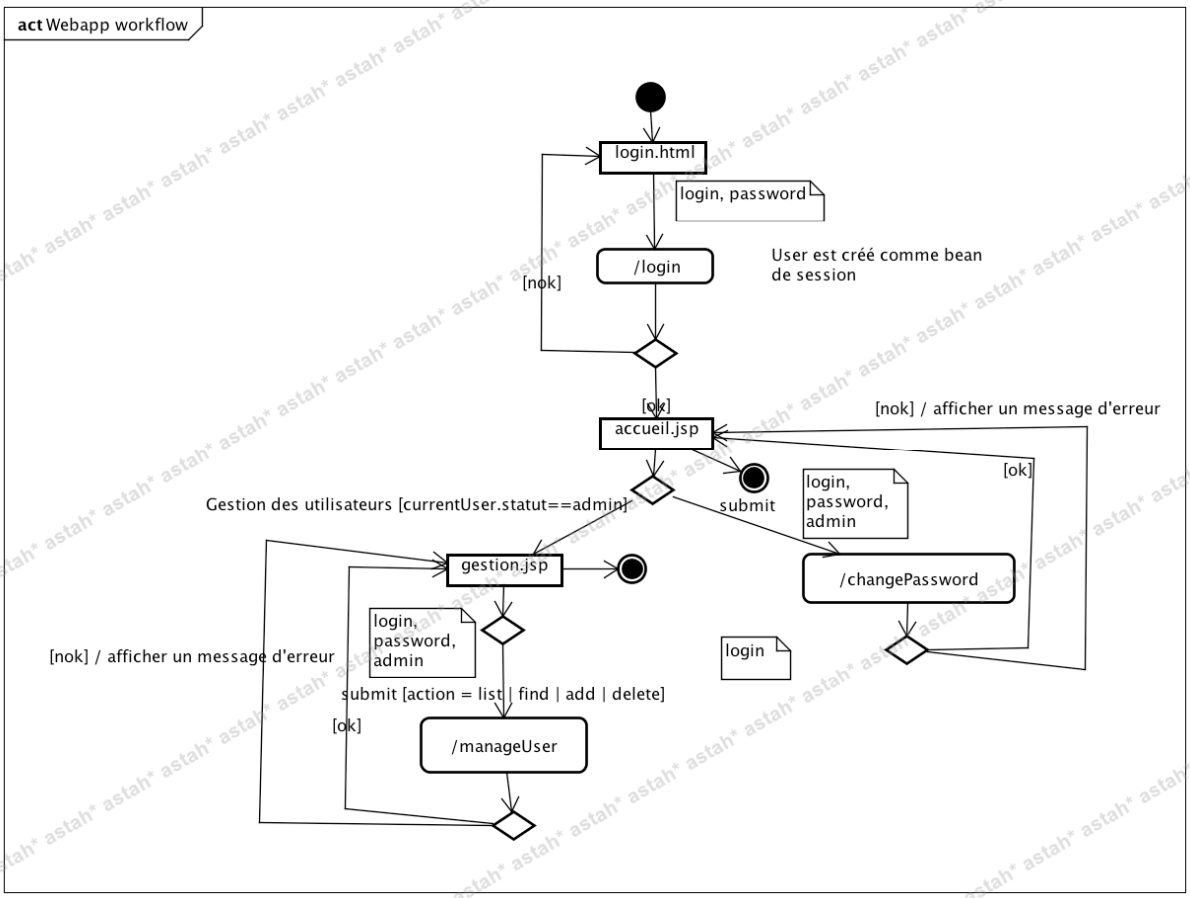

Figure 4. Enchaînement des pages et des actions dans la webapp

http://localhost:1977/TPServlet/login.xhtml
http://localhost:1977/TPServlet/accueil.jsp
http://localhost:1977/TPServlet/gestion.jsp