Laboratoire IBD *No 2*

Laboratoire «Servlets & JSP»

Eric Lefrançois – 21 Octobre 2011

Préambule

1 Objectifs du labo

Cette deuxième partie du laboratoire nous permettra :

- d'apprendre à utiliser les servlets et les JSP au travers de quelques exercices simples (suite de la première partie);
- d'utiliser les possibilités de l'utilitaire «**ant**».

2 Ant

«Ant» est un outils de construction, à l'image de «make» du système Unix. Ant est écrit en Java.

Cet outil est pratiquement indispensable si l'on désire utiliser JBoss de manière efficace.

2.1 INSTALLATION

- Télécharger «Ant» depuis le site http://www.apache.org/ant/
- Au moyen de Unzip, placer les fichiers dans `c:\javatools\ant` (par exemple)
- Ajouter la variable d'environnement **`ANT_HOME`** `= c:\javatools\ant`
- Ajouter le chemin **`ANT_HOME\bin`** dans la variable système `Path`
- Pour tester l'installation, taper «`ant`» dans une fenêtre de commande, le message «`Buildfile: buid.xml does not exist` !» devrait s'afficher.

2.2 UN PREMIER EXEMPLE

Création du fichier `build.xml`

«`build.xml`» est un fichier qui sera utilisé par l'utilitaire ant dans le but de construire et de déployer notre application.

Voici le code d'un petit fichier **build.xml** qui compile un programme source Java situé dans le répertoire **`src`** et place les fichiers résultant de la compilation dans le répertoire **`classes`**.

```
<?xml version="1.0"?>
<!-- Un petit fichier de construction -->
<project name="Ant premier test" default="build" basedir=".">
      <target name="build" >
            <javac srcdir="src" destdir="classes" debug="true"
             includes="**/*.java" />
      </target>
</project>
```

- La première ligne du fichier build.xml représente la déclaration du type de document (un fichier xml)
- La ligne suivante est un commentaire.
- La troisième ligne constitue la balise «`project`». Chaque fichier de construction contient toujours une balise «`project`», et toutes les instructions sont placées à l'intérieur de cette balise.

La balise «project»

```
<project name="Ant premier test" default="build" basedir=".">
```

Cette balise requiert 3 attributs: les attributs **`name`**, **`default`** et **`basedir`**.

Attribut	Description
`name`	Nom du projet
`default`	Nom du target principal par défaut (*default target*) à utiliser lorsqu'aucun target n'est spécifié en ligne de commande: > **`ant`** <*return*> -- *Utilisation du target par défaut* > **`ant`** *nomTarget* -- *Utilisation du target «nomTarget»*
`basedir`	Nom du répertoire de base à partir duquel toutes les constructions de chemins seront opérées.

Tous ces attributs sont requis !

Un même projet peut contenir un ou plusieurs «**targets**»: un «target» regroupe un ensemble de tâches à accomplir. Cet exemple sera limité à un seul «target»: «build», qui utilise **javac** pour compiler les fichiers java.

```
<target name="build" >
      <javac srcdir="src" destdir="classes" debug="true"
       includes="**/*.java" />
</target>
```

Exercice 1 ⇨ Tester le fichier build.xml avec le programme suivant:

```
public class Essai{
      public static void main (String args[]){
            System.out.println("Un essai..");
      }
}
```

2.3 EN SAVOIR PLUS SUR ANT ?

Consulter la documentation Documentation de ant accessible à partir de son répertoire d'installation: C:*rep_installation*\docs\manual\index.html

Notamment, il est dit qu'un *target* peut dépendre de plusieurs targets. On peut par exemple avoir un target pour compiler, et un target pour déployer l'exécutable. Or, on peut déployer l'exécutable uniquement si ce dernier a d'abord été compilé.. Ainsi, le target de déploiement est donc **dépendant** du target de compilation.

Ant essaye d'exécuter les targets indiqués dans l'attribut **`depends`** dans l'ordre dans lequel ils apparaissent (de gauche à droite). Il est possible qu'un target soit déjà exécuté si par exemple un target précédent dépendait déjà de ce dernier.

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="B"/>
<target name="D" depends="C,B,A"/>
```

Supposons que l'on désire exécuter le target D. On pourrait supposer, en observant son attribut de dépendance, que les targets seront exécutés en suivant l'ordre C, puis B, puis A. Il n'en est rien! C dépend de B, et B dépend de A. Ainsi, A sera d'abord exécuté, puis B, puis C, et finalement D.

Exercice 2

En consultant la tâche d'exécution «java», rajouter le target «**exec**» dans le fichier «build.xml» en spécifiant qu'il s'agit du target par défaut et en spécifiant ce dernier avant le target «build». La commande java utilisera notamment l'attribut `classpath` pour spécifier le chemin d'accès (`classes/`).

2.4 DÉVELOPPER UNE APPLICATION WEB AVEC ANT

En premier lieu, nous allons créer un répertoire qui sera dédié au développement de l'application Web.

Ce répertoire obéira à l'architecture suivante (*il s'agit bien sûr d'un exemple, libre à chacun d'adopter son architecture personnelle !*)

Répertoire	Description
`monApplicWeb`	Répertoire de base, contenant: o le fichier build.xml, utilisé par Ant, o le fichier `web.xml`, descripteur de déploiement de notre application o ainsi que l'archive `monApplicWeb.war` qui sera générée par l'utilitaire Ant.
`/build`	Répertoire de base prévu pour contenir tous les fichiers qui seront générés par l'utilitaire Ant. Voir ci-après.
`/build/war`	Répertoire utilisé pour l'assemblage du dossier `MonApplicWeb.war` (avec les sous-répertoires `WEB-INF`, `classes` et `lib`)
`/build/classes`	Pour placer tous les fichiers `.class` résultant des compilations.
`/jar`	Pour placer tous les fichiers jar requis par notre application.
`/src`	Pour placer tous les fichiers source `.java` (servlets et classes utilitaires)
`/web`	Pour placer tous les fichiers `jsp`, `html`, images etc.
`/xml`	Pour placer le descripteur de déploiement de l'archive war (fichier `web.xml`)

Exercice 3

Mettre en place l'architecture suivante:

```
monApplicWeb
monApplicWeb/jar
monApplicWeb/src
monApplicWeb/web
monApplicWeb/xml
```

Le répertoire **build,** ainsi que ses sous-répertoires, seront créés automatiquement par l'utilitaire ant.

Reprendre l'exercice 4 de la première partie du laboratoire (servlet «Recoucou»).

1. Placer le fichier Coucou.java (servlet) dans monApplicWeb/src
2. Placer le descripteur de déploiement web.xml dans le répertoire de base monApplicWeb/xml
3. Placer la librairie servlet (jboss-servlet-api_3.0_spec-1.0.0.Final.jar) dans le répertoire monApplicWeb/jar
4. Placer le fichier **build.xml** (fourni par le professeur et présenté ci-après et ***à ajuster par vos soins***) dans le répertoire de base «**monApplicWeb**»

☞ Exécuter ant et tester le résultat.

Fichier de construction build.xml

Le fichier de construction, présenté ci-dessous, contient un certain nombre de nouvelles balises intéressantes:

- **<property../>** Pour définir des variables
- **<mkdir../>** Pour créer des répertoires
- **<copy/>..</copy>** Pour copier des fichiers
- **<jar../>** Pour créer des archives

```
<?xml version="1.0" ?>
<!-- ................................................ -->
<!-- Fichier de construction pour applications Web       -->
<!-- build.xml, Juin 2011                                -->
<!-- Eric Lefrançois                                     -->
<!-- ................................................ -->

<!-- !!!!! Controler ou ajuster:                          -->
<!-- !!!!! valeur de "warFile" (voir target "init")      -->
<!-- !!!!! valeur de "classpath" (voir target "build")   -->


<project name="Jboss war" default="all" basedir=".">
      <target name="init">
            <!-- !!!!! AJUSTER LA VALEUR DE "warFile" !!!!!!!! -->
            <property name="warFile" value="monApplicWeb.war"/>
            <property name="root" value="${basedir}"/>
            <property name="srcDir" value="${root}/src"/>
            <property name="webDir" value="${root}/web"/>
            <property name="classesDir" value="${root}/build/classes"/>
            <property name="jarDir" value="${root}/jar"/>
```

```
            <property name="xmlDir" value="${root}/xml"/>
            <property name="warDir" value="${root}/build/war"/>

            <!-- Création de WEB-INF et du répertoire de classes -->
            <mkdir dir="${root}/build"/>
            <mkdir dir="${root}/build/war"/>
            <mkdir dir="${root}/build/classes"/>
            <mkdir dir="${warDir}/WEB-INF"/>
            <mkdir dir="${warDir}/WEB-INF/classes"/>
      </target>

      <!-- Target principal -->
      <target name="all" depends="init,build,buildWar"/>

      <!-- Construction -->
      <!-- !!!!! AJUSTER LA VALEUR DE "classpath" !!!!!!!! -->
      <target name="build" >
            <javac srcdir="${srcDir}"
                        Classpath="."
                        destdir="${classesDir}"
                        debug="true" includes="**/*.java" />
      </target>

      <!-- Création de l'archive .war .. -->
      <target name="buildWar" depends="init">
            <copy todir="${warDir}/WEB-INF/classes">
                  <fileset dir="${classesDir}" includes="**/*.class" />
            </copy>

            <copy todir="${warDir}/WEB-INF">
                  <fileset dir="${webDir}" includes="**/*.*" />
            </copy>

            <copy todir="${warDir}/WEB-INF">
                  <fileset dir="${xmlDir}" includes="web.xml" />
            </copy>

            <!-- Création du fichier war -->
            <jar jarfile="${root}/${warFile}" basedir="${warDir}" />
      </target>

</project>
```

Exercice 4

Complétez le fichier `build.xml` de manière à ce que l'archive `monApplicWeb.war` soit directement copiée dans le répertoire de déploiement de JBoss, et propre à être testée.

3 L'API Session

L'API Session est l'un des plus importants pour les servlets, car il permet d'implémenter la notion de session tout au long d'une séance. En effet, HTTP est un protocole sans états, et il n'est donc pas possible de maintenir très simplement des informations d'identification entre deux requêtes HTTP. L'API Session (`HttpSession`) résoud ces problèmes de manière particulièrement élégante, indépendamment de l'utilisation de Cookies.

3.1 IMPLÉMENTATION D'UN COMPTEUR

Exercice 5

Ecrire une servlet qui implémente un compteur de requêtes dans le cadre d'une session donnée.

L'invocation de cette servlet devrait donner quelque chose comme cela:

SessionServlet Output

You have hit this page **7** times.

Exercice 5-bis Votre solution fonctionne-t'elle si vous bloquez l'utilisation des cookies ?

3.2 IMPLÉMENTATION D'UN CADDY

Exercice 6

On trouve souvent, dans les sites dédiés au commerce électronique, la notion de caddy dans lequel on range ses acquisitions virtuelles jusqu'au moment où il faut les payer à la caisse. On propose dans cette manipulation d'implémenter un petit formulaire composé d'un seul champ éditable, qui sera le prix de l'article à acheter, de deux champs non éditables, qui correspondront au prix total des achats déjà emmagasinés dans le caddy et au nombre d'objets contenus dans le caddy, et d'un bouton `Submit`.

On implémentera de plus une servlet qui totalisera les prix envoyés à chaque pression de `Submit`, et qui effectuera la mise à jour du formulaire (prix total et «nombre d'achats»).

A la fin de la session (par exemple caractérisée par un achat de valeur nulle ou négative, ou tout autre mécanisme que vous jugerez judicieux), on aimerait de plus afficher une page HTML contenant un résumé de tous les achats effectués.

4 Java Server Pages

Exercice 7

Rappelons que les JSP permettent de mélanger du code HTML et du code Java plutôt que d'écrire de trop nombreuses instructions `out.println()`.

On propose dans cette dernière manipulation d'implémenter l'exercice précédent (caddy) en utilisant la technologie JSP.

5 Accès à une base de données

Exercice 8

Compléter l'exercice précédent en stockant dans une base de données le résumé de tous les achats effectués par tous les clients du site.

La table obéira au schéma relationnel suivant:

```
Achats(
      IdSession:INTEGER<cléprimaire>
      IdClient: CHAR(2)
      date: DATE
      nombreAchats: INTEGER
      prixTotal: INTEGER
      )
```

Dans le cadre de ce laboratoire, utiliser le driver de type 4 prévu pour Java.