TP JEE (0)
Introduction à MAVEN

Apache Maven est un système très populaire de gestion de construction pour les projets Java, conçu pour supprimer les tâches difficiles du processus de construction. À partir d'une configuration Maven, il est possible de construire des applications sur des types de configurations différentes et avec des IDE différents (Éclipse, Netbeans ...).

Table des matières

1 Préparation aux TD

1.1 Bitbucket

Chaque TP est noté. Vous devez rendre l'ensemble de vos projets avant le TP suivant. Les projets doivent absolument être versionnés sur Bitbucket :

1. créez vous un compte `https://bitbucket.org/account/signup/`
   (a) utilisez un nom d'utilisateur du type *prenom_nom* ou *pnom*.
   (b) ainsi que l'adresse email de l'IUT.
2. allez sur le projet :
   `https://bitbucket.org/jee_dutbx1/2013-groupe-pnom1-pnom2`
3. clicker sur *Fork* :

(a) dans la partie *owner* sélectionnez *jee_ dutbx1*.
(b) dans la partie *nom* remplacez *groupe, pnom1* et *pnom2* par le nom du groupe de TD et le nom des membres de votre équipe.

Sur vos postes de travail

1. éditez le fichier *~/.hgrc* :

```
[ui]
username = Prenom Nom <prenom.nom@monemail.fr>
editor = nano
```

2. placez vous ensuite dans le dossier de votre choix puis *cloner* votre nouveau projet :

```
hg clone <URL de votre projet>
```

3. dans le dossier créé par la commande *hg* vous trouverez une hiérarchie de dossiers et de fichiers à respecter pour répondre aux questions de TD.

Première prise en main

1. modifiez le fichier readme.md en respectant la syntax de type Markdown. [1]
2. *validez* vos changements via la commande *hg commit* et *poussez* les vers bitbucket en utilisant la commande *hg push*.
3. vérifiez que vos changement soient bien présents sur le site web du projet.

Une fois le travail demandé terminé (à la fin de chaque question ou du TD)

1. ajoutez les fichiers au dépôt (ATTENTION : pas de fichiers auto-générés comme le dossier target ou les fichiers war) :

```
hg add monfichier ou hg add mondossier
```

2. *commitez* :

```
hg commit
```

3. *taguez* :

```
hg tag tp0-maven
```

4. et *pushez* :

```
hg push
```

1.2 JBoss (à la maison)

JBoss [2] est un serveur d'application Java développé par Red Hat. Il offre un contexte d'exécution pour des composants applicatifs Java (fichiers war ou ear pour Enterprise ARchive).

Installation (à la maison)

- Vérifiez que Java est bien installé.
- Télécharger *Jboss As 7.1.Final* [3]
- Dézippez le fichier.
- Ajouter une variable d'environnement JBOSS_HOME=/cheminvers/jboss/jboss-as-7.1.1.Final.
- Vous pouvez démarrez JBoss avec la commande : ./bin/standalone.sh

1. http://daringfireball.net/projects/markdown/syntax
2. http://www.jboss.org/jbossas/
3. http://download.jboss.org/jbossas/7.1/jboss-as-7.1.1.Final/jboss-as-7.1.1.Final.zip

Information importantes :

1. À l'IUT, JBoss se trouve dans le dossier /opt.
2. Si vous travaillez à la maison, attention à ce que la variable d'environnement JAVA_HOME pointe bien vers le dossier de votre JVM. Ce chemin dépend de votre système.

2 Création d'un projet simple

2.1 Un premier projet Maven

1. Créez un nouveau projet *Maven*[4] appelé *Contact* qui doit être placé dans un groupe *org.iut.contact* en vous inspirant de la ligne de commande suivante :

```
mvn archetype:generate \
  -DarchetypeGroupId=org.apache.maven.archetypes \
  -DgroupId=com.labri.tulip \
  -DartifactId=Tulip \
  -Dpackagename=com.labri.tulip
```

2. Commentez les arguments de la ligne de commande précédente.
3. Parcourez le dossier qui à été créé. Que contient-il ? Qu'est ce que le fichier pom.xml ? Où est la classe contenant la fonction *main* ?
4. A quoi sert la partie du fichier pom.xml contenant la chaîne de caractère "junit" ?
5. Compilez-le projet (`mvn compile`).
6. A quoi sert le dossier test ? Lancez les tests (`mvn test`).
7. Créez un JAR (`mvn package`). Où est-il généré ?

2.2 Developper avec Eclipse sur un projet Maven

1. Ouvrir Eclipse et conbfigurez les workspace.
2. Configurer le workspace d'Eclipse.

```
mvn -Declipse.workspace=<path-to-eclipse-workspace> eclipse:add-maven-
    repo
```

3. Générez un projet Eclipse avec la commande :

```
mvn eclipse:eclipse -Dwtpversion=2.0
```

4. Importez le projet dans Eclipse.
5. Lancez les tests unitaires dans éclipse.

2.3 Le code métier

1. Dans un package org.iut.contact.model, créez une classe Contact avec les attributs : id(Integer), firstname (String), lastname(String), email(String), phoneNumber(String), birthdate(Date).
2. Faites ajouter par Eclipse un constructeur par défaut, les getters et setters de chaque attribut.
3. N'oubliez pas de redéfinir les méthodes *equals*, *toString* et *hashCode*.
4. Dans un package org.iut.contact.service créez une classe ContactService. Cette classe doit fournir un ensemble de méthodes permettant d'ajouter, supprimer, récupérer (à l'aide d'un identifiant), rechercher (à l'aide d'un nom) ou enregistrer un contact dans une table de hashage (clé=identifiant, valeur=contact)[5]. Ajoutez aussi une méthode Boolean isBirthday(Contact c).

4. http ://maven.apache.org/
5. Javadoc HashMap : `http://docs.oracle.com/javase/6/docs/api/java/util/HashMap.html`

5. Pour chaque méthode, créez plusieurs tests unitaires pour vérifier leurs comportements.
6. Corrigez les méthodes si les tests ne fonctionnent pas.

3 Transformation de Contact en projet Web

3.1 Modification du fichier Maven

1. Modifiez le fichier pom.xml pour changer le type de packaging (jar => war) ?
2. créez un fichier web.xml dans src/main/webapp/WEB-INF/

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="
    http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/
    xml/ns/javaee/web-app_2_5.xsd" xsi:schemaLocation="http://java.sun.
    com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
     version="3.0">
  <display-name>Contact</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

3. Regenerez un projet Eclipse.

```
mvn eclipse:eclipse -Dwtpversion=2.0
```

4. Ouvrez le projet dans Eclipse (*Import existing project*). Si l'erreur suivante apparait,

```
ContactJAR:             Unknown Faceted Project Problem (Java Version
    Mismatch)
```

Ajoutez les lignes de configuration suivantes au fichier pom.xml

```
<build>
  <pluginManagement>
    <plugins>
      <!-- Compiler plugin enforces Java 1.6 compatibility and activates
           annotation processors -->
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.3.2</version>
        <configuration>
          <source>1.6</source>
          <target>1.6</target>
        </configuration>
       </plugin>
    </plugins>
  </pluginManagement>
</build>
```

Puis, mettez à jour le projet Eclipse.

5. Créer un fichier .war à l'aide de Maven.

3.2 Création d'une Servlet (et utilisation du descripteur de déploiement)

1. Créez une nouvelle Servlet *ContactServlet.java* dans un package org.iut.contact.servlet.

```
public class ContactServlet extends HttpServlet {
```

```
        private static final long serialVersionUID = 1L;

    public ContactServlet() {
    }

        protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException,
            IOException {
                // TODO Auto-generated method stub
        }

        protected void doPost(HttpServletRequest request,
            HttpServletResponse response) throws ServletException,
            IOException {
                // TODO Auto-generated method stub
        }
}
```

2. La classe ne compile pas pourquoi ?
3. Ajoutez la dépendance maven suivante :

```
<dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-api</artifactId>
        <version>6.0</version>
</dependency>
```

4. dans le fichier web.xml ajouter un mapping vers la servlet :

```
<servlet>
        <servlet-name>ContactServlet</servlet-name>
        <servlet-class>org.iut.contact.ContactServlet</servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>ContactServlet</servlet-name>
        <url-pattern>/contact</url-pattern>
</servlet-mapping>
```

5. Copiez-collez le code suivant dans la méthode doGet de votre servlet :

```
PrintWriter out = response.getWriter();
out.println( "Hello !" );
out.flush();
out.close();
```

6. Créer un nouveau WAR avec Maven.
7. Démarrer JBOSS (./bin/standalone.sh)
8. Déployer le WAR sur jboss (copier-coller le dans le dossier standalone/deployments/).
9. Testez la nouvelle servlet (`http://localhost:8080/Contact-1.0-SNAPSHOT/contact`)
10. Qu'est ce qu'une Servlet ?
11. Commentez et expliquez le rôle des différentes méthodes de la Servlet.
12. Quel est le rôle du fichier web.xml. Commentez son contenu.
13. Dans la nouvelle Servlet affichez les informations d'un contact et utiliser la classe ContactService afin de déterminer si c'est l'anniversaire de ce même contact.

3.3 Déclaration d'une Servlet via une annotation Java

1. Créez une nouvelle Servlet nommée AllContactsServlet, mais cette fois-ci en utilisant l'annotation Java @WebServlet(name = "....", urlPatterns = "/....") (Ne modifiez pas le fichier

web.xml). L'extrait de code suivant montre un exemple :

```
@WebServlet(name = "AllContactsServlet", urlPatterns = { "/all" })
public class AllContactsServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;

    public AllContactsServlet() {
        super();
    }

        protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException,
            IOException {
//                  ......
        }

}
```

2. Deployez et testez.
3. Dans ce cas précis, à quoi servent les annotations Java ? Quel mécanisme permet au serveur d'application d'interpréter les annotations ?
4. Modifiez la servlet pour afficher l'ensemble des contacts gérés par la classe ContactService.

4 Pour aller plus loin

4.1 Debug depuis Eclipse

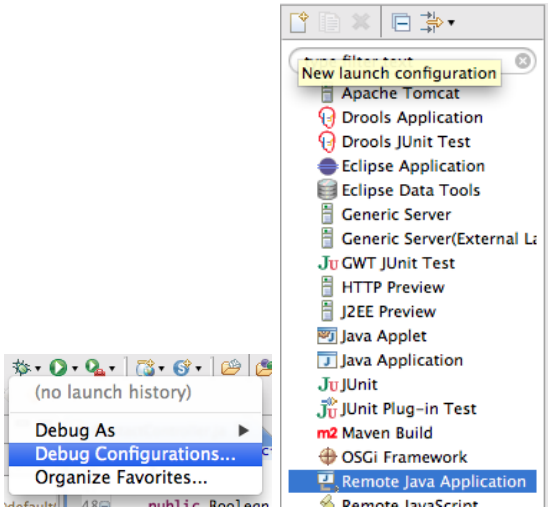

FIGURE 1 – Debug d'une application JEE depuis eclipse.

1. Dans le dossier de JBoss, copiez le fichier /bin/standalone.sh vers /bin/standalone-debug.sh :

```
cp ./bin/standalone.sh ./bin/standalone-debug.sh
```

Puis, ajoutez les lignes suivante en début de fichier.

```
JAVA_OPTS="$JAVA_OPTS -Xdebug -Xrunjdwp:transport=dt_socket,server=y,
    suspend=y,address=8787"
```

2. Dans Eclipse (voir figure 1) :
   (a) Debug Configurations ;
   (b) Créez une nouvelle configuration de type “Remote Java Application”.
   (c) Choisissez le projet ;
   (d) Spécifiez le port 8787 ;
3. Relancer jboss en mode debug ./bin/standalone-debug.sh.
4. Lancer le mode debug dans eclipse.
5. ajoutez un point d'arrêt et vérifier que tout fonctionne bien.

4.2 Téléchargez des exemples de projets JEE

```
git clone git://github.com/jboss-jdf/jboss-as-quickstart.git --branch jdf
    -2.0.0.Final
http://www.jboss.org/jdf/quickstarts/jboss-as-quickstart/
```

4.3 Introduction à Mercurial

Lisez attentivement `http://mercurial.selenic.com/quickstart/` et `http://hgbook.red-bean.com/read/a-tour-of-mercurial-the-basics.html`