

# Middleware – Servlet – MAVEN

29 janvier 2018

# 1

## Création d'une servlet

---

**M**AVEN vous permet de gérer intégralement votre projet sans vous soucier des problèmes de dépendances entre les différentes librairies importées. La particularité de **maven** est de s'appuyer sur un ensemble de conventions pour compiler du code, créer des fichiers JAR, déployer des WAR , etc. On se propose dans la suite d'étudier le fonctionnement de **maven**.

1. Veuillez télécharger **maven** 3.2.1 (<http://maven.apache.org/download.html>)
2. Désarchiver **maven** dans votre *home directory*, mettre à jour la variable PATH de votre *shell*.

```
# export M2_HOME=répertoire_maven
# export PATH=$PATH:$M2_HOME/bin
```

3. Saisir la commande suivante afin de commencer un projet **maven** nommé Tuto1

```
# mvn archetype:generate -DgroupId=net.tuto1.ws.service
-DartifactId=Tuto1 -DarchetypeArtifactId=maven-archetype-webapp
```

4. La structure de votre répertoire devrait avoir la forme suivante :

```
Tuto1
├── pom.xml
├── src
│   ├── main
│   │   ├── java
│   │   ├── resources
│   │   ├── webapp
│   │   └── WEB-INF
└── target
```

- ➡ `pom.xml`, ce fichier décrit en détail votre projet Java
- ➡ `src`, contient les fichiers sources de votre projet
- ➡ `main/java`, contient les fichiers source Java qui seront inclus dans les packages JAR, WAR générés etc...
- ➡ `main/webapp`, contient les pages .jsp
- ➡ `main/WEB-INF`, contient les fichiers de configuration de la servlet
- ➡ `target`, ce répertoire rassemble tout le code généré à partir du code source, bytecode, etc...

Certains répertoires peuvent ne pas être créés comme par exemple `src/main/java`. Dans ce cas, créer les à la main.

5. Editez le fichier `pom.xml` présent à la racine de `Tuto1`. Ajouter les repositories à partir duquel `maven` pourra télécharger les librairies nécessaires pour compiler votre projet.

Listing 1– `pom.xml` ➡

```
1  <project>
    ...
    <repositories>
        <repository>
5      <id>maven2-repository.dev.java.net</id>
        <name>Java.net Repository for Maven 2</name>
        <url>http://download.java.net/maven/2/</url>
        </repository>
    </repositories>
10  ...
    </project>
```

- ➡ Il ne peut y avoir qu'une seule balise xml `<repositories>` dans un fichier `pom.xml`

6. En cas de problèmes avec les dépôts distants, vous pouvez ajouter le dépôt suivant.

Listing 2– pom.xml ➤

```
1  <project>
    ...
    <repositories>
        ...
5   <repository>
        <id>maven2-bromberg.dev.java.net</id>
        <name>Java.net Repository for Maven 2</name>
        <url>http://download.java.net/maven/2/</url>
        </repository>
10  ...
    </repositories>
    ...
</project>
```

## Conception de la servlet

7. Créer la servlet Java suivante dans le répertoire `src/main/java/net/tuto1/ws`

Listing 3– SimpleServlet.java ➤

```
1  package net.tuto1.ws;
    import java.io.*;
    import javax.servlet.*;
    import javax.servlet.http.*;

5   public class SimpleServlet extends HttpServlet {
        public void doGet(HttpServletRequest request,
                           HttpServletResponse response)
            throws ServletException, IOException {

10         PrintWriter out = response.getWriter();
            out.println( "Hello_World\n" );
            out.flush();
            out.close();

15     }
}
```

8. Taper `mvn clean install` ou `mvn compile`. La compilation échoue, il faut rajouter les dépendances avec les API des servlets. Rajouter la dépendance suivante dans votre fichier `pom.xml`

Listing 4– `pom.xml` [download](#)

```
1  <project>
    ...
    <dependencies>
      <dependency>
5    <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
      <scope>provided</scope>
      </dependency>
10 </dependencies>
    ...
  </project>
```

- Il ne peut y avoir qu'une seule balise xml `<dependencies></dependencies>` incluse dans la balise dans un fichier `pom.xml`
- `<scope>provided</scope>` indique à maven que le package manquant ne doit pas être inclut dans le package war correspondant à l'application.

9. Editez le fichier `pom.xml` présent à la racine de *Tuto1*. Indiquer la version du compilateur que vous souhaitez utiliser pour compiler l'ensemble du projet. En particulier, nous choisissons la version 1.5 car nous allons utiliser des annotations dans le code source Java.

Listing 5– `pom.xml` ➤

```
1  <project>
    ...
    <build>
      <plugins>
5   <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.0.2</version>
      <configuration>
10  <source>1.5</source>
      <target>1.5</target>
      </configuration>
      </plugin>
      </plugins>
15 </build>
    ...
  </project>
```

- ▮ Il ne peut y avoir qu'une seule balise xml `<build>` dans un fichier `pom.xml`
- ▮ Il ne peut y avoir qu'une seule balise xml `<plugins></plugins>` incluse dans la balise `<build>` dans un fichier `pom.xml`

# 2

## Déploiement de la Servlet

---

### Configuration

1. Avant de pouvoir déployer votre *servlet*, il faut la configurer afin qu'elle puisse fonctionner dans un conteneur de servlet. Rendez vous dans le répertoire `/src/main/webapp/WEB-INF`. Et éditer le fichier `web.xml` comme suit :

Listing 6– web.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
    <web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
5      http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
        <display-name>Hello world servlet</display-name>
        <servlet>
            <servlet-name>FirstServlet</servlet-name>
            <servlet-class>net.tuto1.ws.SimpleServlet</servlet-class>
10         <load-on-startup>1</load-on-startup>
        </servlet>
        <servlet-mapping>
            <servlet-name>FirstServlet</servlet-name>
            <url-pattern>/test</url-pattern>
15        </servlet-mapping>
    </web-app>
```

---

2. Le fichier `web.xml` est standardisé par la JSR-315. On retiendra les balises suivantes qui sont fondamentales pour le bon fonctionnement de votre *Web Service*.

- ▀ Les balises `<servlet>...</servlet>`, permettent d'indiquer quelle servlet sera chargée par le serveur d'applications. En l'occurrence, on utilise, dans le contexte de ce TD/TP, la pile Web Service de Sun (nom de code Metro). Par conséquent, on utilise la classe `com.sun.xml.ws.transport.http.servlet.WSServlet` qui va s'occuper de réceptionner toutes les requêtes HTTP.
- ▀ Les balises `<servlet-mapping>...</servlet-mapping>`, permettent de spécifier quelle servlet sera exécutée suivant l'URL demandée par le client HTTP.
- ▀ Les balises `<url-pattern>...</url-pattern>`, permettent de définir un motif qui déclenchera la redirection de la requête http vers la servlet adéquate. Par exemple, `<url-pattern>/services/*</url-pattern>` va mapper toutes les requêtes ayant dans l'URL demandée le motif `/services/` vers la servlet ayant le nom `WebServicePort`.

3. Modifier le fichier `pom.xml` à la racine de votre répertoire `Tuto1`, pour indiquer quel conteneur de servlet vous souhaitez utiliser. Dans un premier temps, nous allons utiliser le conteneur Jetty.

Listing 7– pom.xml ➤

```

1  <project>
    ...
    <build>
        <plugins>
5         ...
        <plugin>
            <groupId>org.mortbay.jetty</groupId>
            <artifactId>jetty-maven-plugin</artifactId>
            <version>8.1.5.v20120716</version>
10        </plugin>
        ...
        </plugins>
    </build>
    ...
15 </project>
```

<http://www.oracle.com/technetwork/java/javaee/servlet/index.html>

- 
4. Une fois le plugin **Jetty** configuré, vous pouvez lancer votre application web grâce à la commande `mvn jetty:run` dans votre *shell*. Pour plus d'informations sur le fonctionnement du plugin, se rendre à l'url suivante [http://wiki.eclipse.org/Jetty/Feature/Jetty\\_Maven\\_Plugin](http://wiki.eclipse.org/Jetty/Feature/Jetty_Maven_Plugin).
  5. Vous pouvez maintenant surfer sur la page `http ://localhost :8080/Tuto1/test`



# 3

## L'API servlet 3.1

La nouvelle api des servlets, à savoir la 3.1 ajoute un certain nombre de nouvelles fonctionnalités. En particulier, le support pour :

- ➡ L'adjonction de filtres
- ➡ L'utilisation de fragments afin de modulariser le code.
- ➡ L'utilisation d'annotations afin de faciliter la programmation et de ne plus avoir à écrire le fichier `web.xml`
- ➡ L'exécution asynchrone

On s'intéresse principalement à l'utilisation des annotations, afin de ne plus à avoir à gérer manuellement le fichier `web.xml`. En particulier, nous allons utiliser l'annotation `@WebServlet` afin de peupler automatiquement le fichier `web.xml`.

Listing 8– SimpleServlet.java avec annotations 📄

```
1 package net.tuto2.ws;
  import java.io.*;
  import javax.servlet.*;
  import javax.servlet.http.*;
5  import javax.servlet.annotation.WebServlet;

  @WebServlet(name="FirstServlet", ❶
              urlPatterns={"/test"}, ❷
              loadOnStartup=1) ❸
10 public class SimpleServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {

15        PrintWriter out = response.getWriter();
        out.println( "Hello World from annotated servlet\n"
                      );
        out.flush();
        out.close();

    }
20 }
```

Le champ **name** ❶ permet d'indiquer le nom de la servlet et remplace la nécessité d'ajouter les balises `<servlet-name>...</servlet-name>` du fichier `web.xml`. Pour spécifier à partir de quelle URL la servlet sera accessible, on utilise le champ **urlPatterns** ❷ qui va remplacer les balises suivantes :

Listing 9– web.xml ➤

```
1  <servlet-mapping>
    <servlet-name>FirstServlet</servlet-name>
    <url-pattern>/test</url-pattern>
  </servlet-mapping>
```

Enfin, le champ **loadOnStartup** ❸ va remplacer avantageusement la balise XML `<load-on-startup>1</load-on-startup>`

Grâce à l'utilisation de l'API 3.1 le fichier `web.xml` devient optionnel. On peut donc le supprimer du répertoire `src/main/webapp/WEB-INF`.

Toutefois, lors de la recompilation du projet *via* la commande `mvn clean install`, maven se plaint de ne pas trouver le fichier `web.xml`. On obtient le message suivant : `Error assembling WAR: webxml attribute is required (or pre-existing WEB-INF/web.xml if executing in update mode) -> [Help 1]`. Pour ne plus avoir cette erreur, il faut configurer le plugin `maven-war-plugin` pour lui indiquer de ne pas vérifier la présence de ce fichier.

Listing 10– pom.xml ➤

```
1  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-war-plugin</artifactId>
    <configuration>
5     <failOnMissingWebXml>>false</failOnMissingWebXml>
    </configuration>
  </plugin>
```