Application Java Servlet avec Maven

Voici un guide complet pour créer et déployer une application Java Servlet simple sur WebSphere Liberty Profile (WLP) en utilisant Maven pour construire le projet et le plugin Maven Liberty pour le déploiement. Cet exemple crée un Servlet de base qui renvoie “Hello World!”lorsqu’il est accédé, le construit dans un fichier WAR avec Maven, et le déploie sur WLP. Nous supposerons que vous avez déjà configuré WLP avec la fonctionnalité `servlet-4.0` activée, car elle fournit le support Servlet nécessaire.

Guide étape par étape

1. Créer la structure du projet Maven

Commencez par configurer une structure d’application web standard Maven. Vous pouvez créer cela manuellement ou utiliser Maven pour la générer.

- **Structure des répertoires :**

```
SimpleServletApp/
  pom.xml
  src/
      main/
          java/
              com/
                  example/
                      HelloServlet.java
          webapp/
              WEB-INF/
                  web.xml
```

- **Optionnellement générer avec Maven :** Exécutez cette commande pour créer la structure, puis ajustez selon vos besoins :

```
mvn archetype:generate -DgroupId=com.example -DartifactId=simple-servlet-app -DarchetypeArtifactId=maven-
```

  Cela crée une structure de webapp de base, que vous modifierez dans les étapes suivantes.

2. Écrire le code du Servlet

Créez un fichier nommé `HelloServlet.java` dans `src/main/java/com/example/` avec le contenu suivant :

```java
package com.example;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class HelloServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException {
        resp.setContentType("text/plain");
        resp.getWriter().write("Hello World!");
    }
}
```

- **Explication :** Ce Servlet répond aux requêtes HTTP GET avec “Hello World!”en texte brut. Il utilise une méthode `doGet` simple et évite les annotations pour la compatibilité avec la configuration explicite de `web.xml`.

3. Créer le descripteur de déploiement `web.xml`

Créez un fichier nommé `web.xml` dans `src/main/webapp/WEB-INF/` avec le contenu suivant :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0
         version="4.0">
    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>com.example.HelloServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
</web-app>
```

- **Explication :** Le fichier `web.xml` définit la classe `HelloServlet` et la mappe au motif d’URL `/hello`. Cela est nécessaire puisque nous n’utilisons pas les annotations `@WebServlet`.

4. Configurer le `pom.xml` Maven

Créez ou mettez à jour `pom.xml` dans le répertoire `SimpleServletApp/` avec le contenu suivant :

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>simple-servlet-app</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- Servlet API (fournie par WLP) -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>4.0.1</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- Plugin Maven WAR pour construire le fichier WAR -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.3.1</version>
                <configuration>
                    <finalName>myapp</finalName>
                </configuration>
            </plugin>
            <!-- Plugin Maven Liberty pour le déploiement -->
```

```
            <plugin>
                <groupId>io.openliberty.tools</groupId>
                <artifactId>liberty-maven-plugin</artifactId>
                <version>3.3.4</version>
                <configuration>
                    <installDirectory>/opt/ibm/wlp</installDirectory>
                    <serverName>myServer</serverName>
                    <appsDirectory>dropins</appsDirectory>
                    <looseApplication>false</looseApplication>
                    <stripVersion>true</stripVersion>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

- **Explication :**
  - **Coordonnées :** Définit le projet avec `groupId`, `artifactId` et `version`. Le `packaging` est défini sur `war` pour une application web.
  - **Propriétés :** Définit Java 8 comme version source et cible.
  - **Dépendances :** Inclut l'API Servlet avec un scope `provided`, car elle est fournie par WLP en temps d'exécution.
  - **Plugin Maven WAR :** Configure le nom du fichier WAR à `myapp.war` en utilisant `<finalName>`.
  - **Plugin Maven Liberty :** Configure le déploiement vers un serveur Liberty à `/opt/ibm/wlp`, nom du serveur `myServer`, déploiement vers le répertoire `dropins`.

5. Construire le projet

Depuis le répertoire `SimpleServletApp/`, construisez le fichier WAR en utilisant Maven :

```
mvn clean package
```

- **Résultat :** Cela compile le Servlet, le regroupe avec `web.xml` dans `target/myapp.war`, et le prépare pour le déploiement.

6. Déployer et exécuter sur WebSphere Liberty

Assurez-vous que votre serveur Liberty (`myServer`) est configuré avec la fonctionnalité `servlet-4.0` activée. Vérifiez votre `server.xml` pour :

```
<featureManager>
    <feature>servlet-4.0</feature>
</featureManager>
```

Déployez et exécutez l'application en utilisant le plugin Maven Liberty :

```
mvn liberty:run
```

- **Ce qui se passe :**
  - Démarre le serveur Liberty en premier plan (s'il n'est pas déjà en cours d'exécution).
  - Déploie `myapp.war` dans le répertoire `dropins` automatiquement.
  - Garde le serveur en cours d'exécution jusqu'à ce qu'il soit arrêté.
- **Vérifier le déploiement :** Recherchez un message de journalisation comme :

  ```
  [AUDIT   ] CWWKT0016I: Web application available (default_host): http://localhost:9080/myapp/
  ```

  Les journaux sont généralement dans `/opt/ibm/wlp/usr/servers/myServer/logs/console.log`.

7. Accéder à l'application

Ouvrez un navigateur et accédez à :

```
http://localhost:9080/myapp/hello
```

- **Sortie attendue :**

  ```
  Hello World!
  ```

- **Découpage de l'URL :**
  - `9080` : Port HTTP par défaut pour WLP.
  - `/myapp` : Racine du contexte à partir du nom du fichier WAR (`myapp.war`).
  - `/hello` : Motif d'URL de `web.xml`.

8. Arrêter le serveur

Puisque `mvn liberty:run` exécute le serveur en premier plan, arrêtez-le en appuyant sur `Ctrl+C` dans le terminal.

Notes

- **Prérequis :**
  - Maven doit être installé et configuré sur votre système.
  - Liberty doit être installé à `/opt/ibm/wlp`, et l'instance de serveur `myServer` doit exister. Ajustez `installDirectory` et `serverName` dans `pom.xml` si votre configuration diffère (par exemple, `/usr/local/wlp` ou `defaultServer`).
  - La fonctionnalité `servlet-4.0` doit être activée dans `server.xml`.
- **Déploiement alternatif :**
  - Pour construire et déployer séparément :

    ```
    mvn clean package
    mvn liberty:deploy
    ```

    Démarrez le serveur manuellement si nécessaire :

    ```
    /opt/ibm/wlp/bin/server start myServer
    ```
- **Configuration du port :** Si votre serveur Liberty utilise un port HTTP différent, vérifiez `server.xml` pour `<httpEndpoint>` et ajustez l'URL en conséquence.
- **Racine du contexte :** Renommez `<finalName>` dans `pom.xml` (par exemple, `<finalName>app</finalName>`) pour changer la racine du contexte en `/app`.
- **Dépannage :**
  - Si le déploiement échoue, vérifiez `installDirectory` et `serverName` dans `pom.xml`.
  - Vérifiez les journaux à `/opt/ibm/wlp/usr/servers/myServer/logs/messages.log` pour les erreurs.
  - Assurez-vous que Java 8 ou une version ultérieure est utilisée, correspondant aux paramètres `maven.compiler`.

Résumé

Ce guide démontre comment : 1. Configurer un projet Maven avec un Servlet simple (`HelloServlet.java`) et `web.xml`. 2. Configurer `pom.xml` avec l'API Servlet, le plugin Maven WAR et le plugin Maven Liberty. 3. Construire l'application en `myapp.war` en utilisant `mvn clean package`. 4. Déployer et exécuter sur WLP en utilisant `mvn liberty:run`. 5. Accéder à "Hello World!"à `http://localhost:9080/myapp/hello`.

Cela fournit une approche simplifiée, basée sur Maven, pour développer et déployer une application Servlet sur WebSphere Liberty Profile.