Aplicación de Servlet de Java con Maven

Aquí tienes una guía completa para crear y desplegar una aplicación simple de Servlet de Java en WebSphere Liberty Profile (WLP) utilizando Maven para construir el proyecto y el plugin de Maven de Liberty para el despliegue. Este ejemplo crea un Servlet básico que devuelve "Hello World!"cuando se accede a él, lo construye en un archivo WAR con Maven y lo despliega en WLP. Supongamos que ya has configurado WLP con la característica `servlet-4.0` habilitada, ya que esto proporciona el soporte de Servlet necesario.

Guía Paso a Paso

1. Crear la Estructura del Proyecto Maven

Comienza configurando una estructura estándar de aplicación web de Maven. Puedes crear esto manualmente o usar Maven para generarlo.

- **Estructura de Directorios:**

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

- **Opcionalmente Generar con Maven:** Ejecuta este comando para crear la estructura, luego ajusta según sea necesario:

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

  Esto crea una estructura básica de webapp, que modificarás en los siguientes pasos.

2. Escribir el Código del Servlet

Crea un archivo llamado `HelloServlet.java` en `src/main/java/com/example/` con el siguiente contenido:

```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!");
    }
}
```

- **Explicación:** Este Servlet responde a las solicitudes HTTP GET con “Hello World!”en texto plano. Utiliza un simple método `doGet` y evita las anotaciones para la compatibilidad con la configuración explícita de `web.xml`.

3. Crear el Descriptor de Despliegue `web.xml`

Crea un archivo llamado `web.xml` en `src/main/webapp/WEB-INF/` con el siguiente contenido:

```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>
```

- **Explicación:** El archivo `web.xml` define la clase `HelloServlet` y la mappa al patrón de URL `/hello`. Esto es necesario ya que no estamos utilizando anotaciones `@WebServlet`.

4. Configurar el `pom.xml` de Maven

Crea o actualiza `pom.xml` en el directorio `SimpleServletApp/` con el siguiente contenido:

```
<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 (proporcionado por 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 para construir el archivo 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 de Liberty para el despliegue -->
```

```
            <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>
```

- **Explicación:**
  - **Coordenadas:** Define el proyecto con `groupId`, `artifactId` y `version`. El `packaging` se establece en `war` para una aplicación web.
  - **Propiedades:** Establece Java 8 como la versión de origen y destino.
  - **Dependencias:** Incluye la API de Servlet con el alcance `provided`, ya que se proporciona por WLP en tiempo de ejecución.
  - **Plugin Maven WAR:** Configura el nombre del archivo WAR a `myapp.war` usando `<finalName>`.
  - **Plugin Maven de Liberty:** Configura el despliegue en un servidor Liberty en `/opt/ibm/wlp`, nombre del servidor `myServer`, desplegando en el directorio `dropins`.

5. Construir el Proyecto

Desde el directorio `SimpleServletApp/`, construye el archivo WAR usando Maven:

```
mvn clean package
```

- **Resultado:** Esto compila el Servlet, lo empaqueta con `web.xml` en `target/myapp.war` y lo prepara para el despliegue.

6. Desplegar y Ejecutar en WebSphere Liberty

Asegúrate de que tu servidor Liberty (`myServer`) esté configurado con la característica `servlet-4.0` habilitada. Verifica tu `server.xml` para:

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

Despliega y ejecuta la aplicación usando el plugin Maven de Liberty:

```
mvn liberty:run
```

- **Qué Sucede:**
  - Inicia el servidor Liberty en primer plano (si no está ya en ejecución).
  - Despliega `myapp.war` en el directorio `dropins` automáticamente.
  - Mantiene el servidor en ejecución hasta que se detenga.
- **Verificar el Despliegue:** Busca un mensaje de registro como:

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

  Los registros suelen estar en `/opt/ibm/wlp/usr/servers/myServer/logs/console.log`.

7. Acceder a la Aplicación

Abre un navegador y navega a:

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

- **Salida Esperada:**

  ```
  Hello World!
  ```

- **Desglose de la URL:**
  - `9080`: Puerto HTTP predeterminado para WLP.
  - `/myapp`: Raíz del contexto del archivo WAR (`myapp.war`).
  - `/hello`: Patrón de URL de `web.xml`.

8. Detener el Servidor

Dado que `mvn liberty:run` ejecuta el servidor en primer plano, deténlo presionando `Ctrl+C` en la terminal.

Notas

- **Prerrequisitos:**
  - Maven debe estar instalado y configurado en tu sistema.
  - Liberty debe estar instalado en `/opt/ibm/wlp`, y la instancia del servidor `myServer` debe existir. Ajusta `installDirectory` y `serverName` en `pom.xml` si tu configuración es diferente (por ejemplo, `/usr/local/wlp` o `defaultServer`).
  - La característica `servlet-4.0` debe estar habilitada en `server.xml`.
- **Despliegue Alternativo:**
  - Para construir y desplegar por separado:

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

    Inicia el servidor manualmente si es necesario:

    ```
    /opt/ibm/wlp/bin/server start myServer
    ```
- **Configuración del Puerto:** Si tu servidor Liberty usa un puerto HTTP diferente, verifica `server.xml` para `<httpEndpoint>` y ajusta la URL en consecuencia.
- **Raíz del Contexto:** Renombra `<finalName>` en `pom.xml` (por ejemplo, `<finalName>app</finalName>`) para cambiar la raíz del contexto a `/app`.
- **Solución de Problemas:**
  - Si el despliegue falla, verifica `installDirectory` y `serverName` en `pom.xml`.
  - Verifica los registros en `/opt/ibm/wlp/usr/servers/myServer/logs/messages.log` para errores.
  - Asegúrate de usar Java 8 o posterior, coincidiendo con las configuraciones de `maven.compiler`.

Resumen

Esta guía demuestra cómo: 1. Configurar un proyecto Maven con un Servlet simple (`HelloServlet.java`) y `web.xml`. 2. Configurar `pom.xml` con la API de Servlet, el plugin Maven WAR y el plugin Maven de Liberty. 3. Construir la aplicación en `myapp.war` usando `mvn clean package`. 4. Desplegar y ejecutarla en WLP usando `mvn liberty:run`. 5. Acceder a “Hello World!”en `http://localhost:9080/myapp/hello`.

Esto proporciona un enfoque basado en Maven para desarrollar y desplegar una aplicación de Servlet en WebSphere Liberty Profile.