Anatomie d'une application Web en Java

Ce didacticiel[1] examine de plus près la structure d'une application Web. Vous allez travailler avec cette structure dans beaucoup de projets Web, alors assurez-vous de bien la comprendre avant de continuer.

1. Répertoire des applications Web

Ce didacticiel suppose que vous utilisez soit un serveur **Jetty** avec son propre `jetty-base` répertoire, soit un serveur **Tomcat**. Ces deux serveurs contiennent un `webapps` répertoire, où ira votre code.

Exemple de projet

Vous pouvez voir un exemple de projet qui utilise la ligne de commande pour empaqueter une application Web ici , et vous pouvez le télécharger en tant que `.zip` fichier ici .

Cette section du didacticiel décrit cet exemple de projet.

2. Servlets

Vous en apprendrez plus sur les servlets dans le didacticiel sur les servlets, mais pour l'instant, tout ce que vous devez savoir est qu'une servlet est une classe Java qui exécute du code sur un serveur lorsque l'utilisateur accède à une certaine URL. Par exemple, voici une classe de servlet hello world :

```
package io.happycoding;

import java.io.IOException;

import jakarta.servlet.annotation.WebServlet;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/hello")
```

---

[1] *happycoding.io*

- *tutorialspoint.com/restful*
- *https://restfulapi.net*
- *oracle-base.com/articles/misc/*

```
public class HelloWorldServlet extends HttpServlet {

  @Override

  public void doGet(HttpServletRequest request, HttpServletResponse response)

      throws IOException {

    response.setContentType("text/html;");

    response.getWriter().println("<h1>Hello world!</h1>");

  }

}
```

Cette servlet utilise l'annotation `@WebServlet` pour mapper l'URL `/hello` et écrit un message `<h1>Hello world!</h1>` codé en dur dans la réponse.

Pour inclure cette servlet dans votre application Web, vous devez la compiler et placer le fichier `.class` généré dans un répertoire spécifique. Continuez à lire pour en savoir plus!

3. Anatomie d'une application Web

Chaque application Web est un dossier qui contient des fichiers dans une certaine structure de répertoires. La structure devrait être comme ceci :

- `Web App Name`
  - Fichiers statiques comme `index.html`, `script.js`, et `styles.css`
  - `WEB-INF/`
    - `classes`
      - Les fichiers `.class` de serveur compilés

Par exemple, vous pourriez avoir un répertoire `HelloWorld` d'applications Web qui ressemble à ceci :

- `HelloWorld`
  - `index.html`
  - `WEB-INF/`
    - `classes`
      - `/io/happycoding/servlets/`
        - `HelloWorldServlet.class`

N'oubliez pas que vous pouvez également utiliser `root` comme nom de l'application Web pour en faire l'application Web de « niveau supérieur » sans inclure son nom dans son URL.

4. En ligne de commande

Je ne recommande pas d'essayer de créer votre code directement à partir de la ligne de commande, mais le faire de cette façon au début peut vous aider à comprendre comment tout s'emboîte.

Vous pouvez créer manuellement la structure de répertoires ci-dessus en créant des dossiers et en y copiant des fichiers. Mais comment compilez-vous vos classes de servlet ? La classe `HelloWorldServlet` référence quelques classes dans le package `jakarta`.

Jetty et Tomcat sont tous deux fournis avec un répertoire `lib` contenant tous les fichiers `.jar` dont vous avez besoin pour créer le code de votre serveur, y compris le fichier jar qui contient le package `jakarta`. Pour compiler vos classes de serveur, vous utiliseriez ce répertoire `lib` comme chemin de classe, en utilisant l'argument `-cp`.

Vous pouvez compiler votre code serveur avec l'une de ces commandes, selon que vous utilisez Jetty ou Tomcat :

Sur Jetty:

- **javac -cp /path/to/jetty/lib/* /path/to/your/code/YourClass.java**
- **javac -cp C:/Users/kevin/Desktop/jetty-home-11.0.5\lib* io/happycoding/servlets/HelloWorldServlet.java**

Sur Tomcat:

- **javac -cp /path/to/tomcat/lib/* /path/to/your/code/YourClass/java**
- **javac -cp C:\Users\kevin\Desktop\apache-tomcat-10.0.7\lib* io/happycoding/servlets/HelloWorldServlet.java**

Dans les deux cas, vous devriez maintenant avoir un fichier `.class` pour la classe que vous avez compilée, que vous pouvez inclure dans la structure de répertoires de l'application Web décrite ci-dessus.

Vous pouvez maintenant copier l'intégralité de votre répertoire d'applications Web dans le répertoire `webapps` de votre serveur, exécuter votre serveur, puis accéder à http://localhost:8080/YourWebAppName/ pour afficher votre application Web !

5. Fichiers .war

Vous savez maintenant comment créer un répertoire d'applications Web, qui contient tous les fichiers dont vous avez besoin pour servir votre site Web, y compris les fichiers HTML et les classes Java qui s'exécutent sur le serveur.

Une autre façon courante de packager votre application Web consiste à créer un fichier .war, qui ressemble beaucoup à un fichier .zip ou à un fichier .jar. Un fichier .war est un **W**eb **A**pplication **R**esource qui recueille un répertoire d'applications Web entier dans un seul fichier. C'est pratique pour le déploiement sur des serveurs en direct, car vous n'avez besoin de télécharger qu'un seul fichier .war au lieu d'un répertoire entier.

Pour créer un fichier .war, cd dans le répertoire de votre application Web (après avoir compilé toutes vos classes), puis exécutez cette commande :

- **jar -cfv YourWebAppName.war** *

Cela devrait vous donner un fichier YourWebappName.war. Vous pouvez déplacer ce fichier dans le répertoire webapps de votre serveur, tout comme vous pourriez déplacer tout le répertoire. Votre serveur va extraire automatiquement votre application Web dans son propre répertoire.

Le format .war est couramment utilisé par de nombreux serveurs. Si vous pouvez créer un fichier .war, vous pouvez déployer un serveur en direct dans des endroits comme Amazon Web Services.