Servlets

Ce didacticiel présente les **servlets** qui vous permettent d'écrire du code Java qui s'exécute sur votre serveur pour répondre aux demandes.

1. Classes de servlets

Une servlet est une classe Java qui exécute certaines fonctions lorsqu'un utilisateur demande une URL à un serveur. Ces fonctions contiennent du code qui réagit aux actions d'un utilisateur et peuvent faire des choses comme enregistrer des données dans une base de données, exécuter une logique et renvoyer les informations nécessaires pour afficher une page.

Pour créer un servlet, créez une classe qui étend la classe `HttpServlet`, puis définissez une fonction de gestionnaire de requêtes. Vous découvrirez les différents types de gestionnaires de requêtes plus tard, mais pour commencer, définissez une fonction `doGet()` :

```
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")
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>");
  }
}
```

Ce code définit une classe `HelloWorldServlet`, qui étend la classe `HttpServlet`. Il remplace la fonction `doGet()`, qui prend `HttpServletRequest` et `HttpServletResponse` comme paramètres. À l'intérieur de `doGet()`, le code obtient un `PrintWriter` de la réponse et y renvoie du contenu HTML.

2. Compilation de classes de servlet

Comme tout autre fichier Java, vous devez compiler votre servlet dans un fichier .class. La seule astuce est que nous devons nous assurer que l'API de servlet Java est sur notre chemin de classe.

3. Annotations

La classe HelloWorldServlet n'a pas de fonction main(), alors comment l'exécuter ? La réponse est que non ! Au lieu de cela, vous devez dire à votre serveur d'exécuter le code lorsqu'un utilisateur lui demande une certaine URL. Pour ce faire, le code utilise l'annotation @WebServlet.

```
...
@WebServlet("/hello")
public class HelloWorldServlet extends HttpServlet {
...
```

Cette annotation indique à votre serveur que cette classe de servlet gérera toutes les requêtes envoyées à l'URL /hello.

Jusqu'à présent, vous n'avez vu que des requêtes GET déclenchées lorsque vous accédez à une URL dans un navigateur Web. Lorsque votre serveur reçoit une requête GET pour l'URL /hello, il appelle automatiquement la fonction doGet() de la servlet avec l'annotation correspondante.

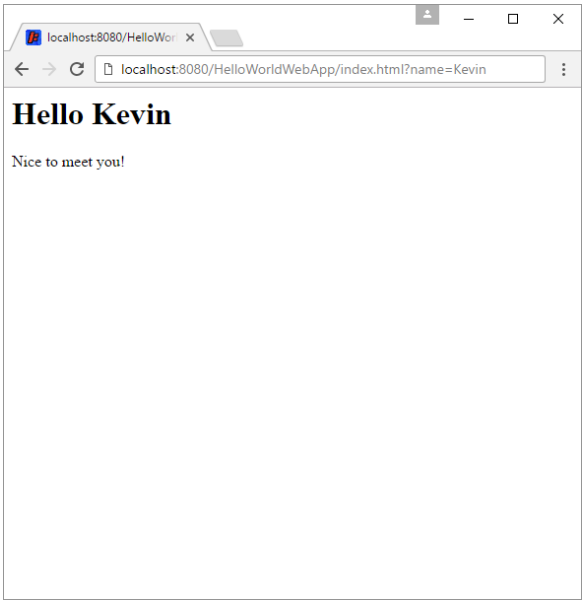
4. Paramètres de requête

Jusqu'à présent, vous avez mappé une seule URL à un servlet. Mais vous pouvez faire des requêtes plus avancées. Par exemple, vous pouvez lire **les paramètres de requête** dans votre servlet et les utiliser dans votre code Java. Pour ce faire, utilisez la fonction request.getParameter(), comme ceci :

```
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/hello")
public class HelloWorldServlet extends HttpServlet {

    @Override
```

```
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {

        String name = request.getParameter("name");

        PrintWriter out = response.getWriter();
        out.println("<h1>Hello " + name + "</h1>");
        out.println("<p>Nice to meet you!</p>");
    }
}
```

Cette servlet utilise la fonction `request.getParameter()` pour obtenir le paramètre `name`, et elle l'utilise pour afficher un message d'accueil en HTML. Ajoutez ce code à votre servlet, puis visitez `/hello?name=Kevin` pour voir ceci :



Essayez de changer le paramètre de requête pour voir des noms différents !

5. Chemins d'URL

N'oubliez pas que l'annotation `@WebServlet` sur une classe de servlet indique à votre serveur à quelle URL mapper le servlet. Par exemple, cette annotation mappe l'URL `index.html` vers une servlet

- @WebServlet("/index.html")

En plus des URL uniques, vous pouvez également utiliser le *symbole générique astérisque pour mapper plusieurs URL sur le même servlet. Voici un exemple :

- @WebServlet("/hello/*")

Cette annotation mappe toute URL commençant par /hello/ vers la servlet. Ainsi, n'importe laquelle de ces URL déclencherait les fonctions de réponse de la servlet :

- /HelloWorldWebApp/hello/
- /hello/Ada
- /hello/Grace/Hopper

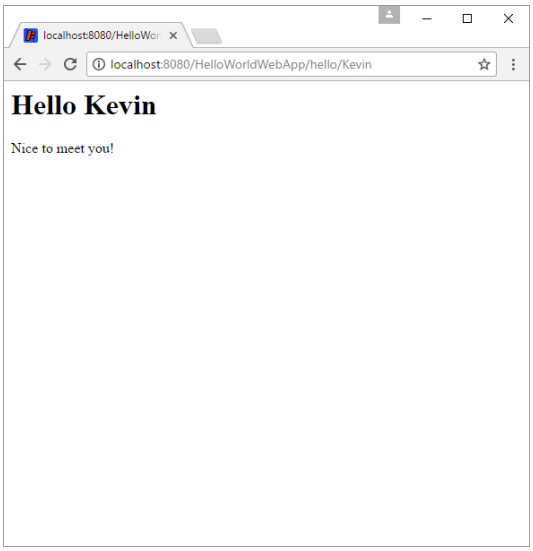
Cela vous permet de générer un contenu différent en fonction de l'URL. Dans votre servlet, vous pouvez utiliser la fonction request.getRequestURI(), qui vous donne l'URL après le domaine (le http://localhost:8080 ou la partie example.com).

Donc, si vous visitez /hello/Kevin, vous pouvez obtenir la partie /hello/Kevin de l'URL en utilisant la request.getRequestURI()fonction. Ensuite, pour obtenir la pièce Kevin, vous pouvez utiliser la fonction substring(). En rassemblant tout ça, ça ressemble à ça :

```
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/hello/*")
public class HelloWorldServlet extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
    IOException {

        String requestUrl = request.getRequestURI();
        String name = requestUrl.substring("/hello/".length());

        PrintWriter out = response.getWriter();
```

```
            out.println("<h1>Hello " + name + "</h1>");
            out.println("<p>Nice to meet you!</p>");
        }
    }
```

Ensuite, visitez `/hello/Kevin` et vous devriez voir ceci:



Essayez maintenant d'accéder à différentes URL pour modifier le message d'accueil. Imaginez à quel point ce serait difficile avec des fichiers HTML statiques !

6. Plusieurs servlets

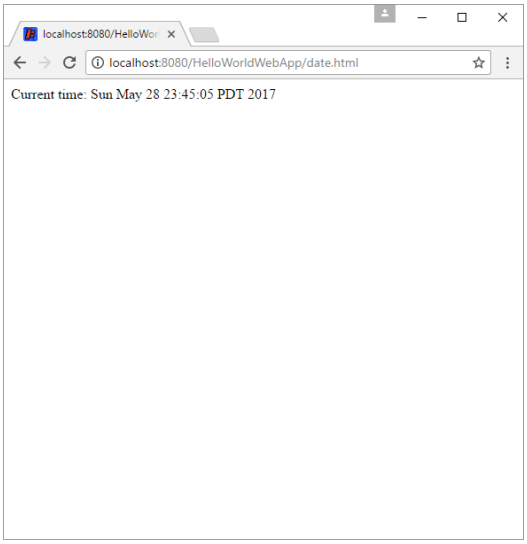
Jusqu'à présent, tous les exemples d'applications Web contenaient un seul servlet. Mais vous n'êtes pas limité à n'utiliser qu'une seule servlet à la fois. Votre application Web peut contenir plusieurs servlets !

Ajoutez une autre servlet à votre application Web en créant une autre classe qui étend la classe `HttpServlet` :

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/date.html")
```

```
public class DateServlet extends HttpServlet {

        @Override
        public void doGet(HttpServletRequest request, HttpServletResponse response) throws
        IOException {

                Date now = new Date();

                PrintWriter out = response.getWriter();
                out.println("<p>Current time: " + now.toString() + "</p>");
        }
}
```

Compilez ce code de la même manière que vous avez compilé votre première classe de servlet, puis ouvrez un navigateur `/date.html` et vous devriez voir ceci :



Vous pouvez également toujours visiter des URL comme `/hello/Felicia` pour déclencher votre première servlet.

La plupart des applications Web se composent de plusieurs servlets, ce qui est un moyen pratique de diviser votre logique.

7. Mélange de contenu statique et dynamique

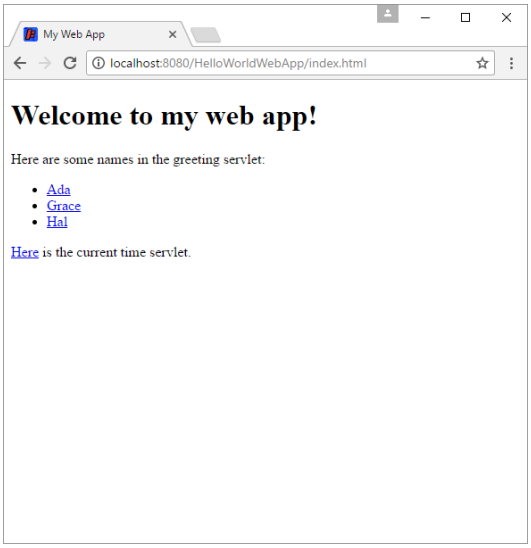
Dans les didacticiels d'installation, vous avez appris à servir des fichiers statiques à partir de votre serveur. Ce tutoriel vous a montré comment créer du contenu dynamique à l'aide d'une classe de servlet. Mais ce n'est

pas un choix : vous pouvez servir à la fois des fichiers statiques et du contenu dynamique à partir du même serveur !

Par exemple, ajoutez cette page index.html statique à votre application Web :

```
<!DOCTYPE html>
<html>
    <head>
        <title>My Web App</title>
    </head>
    <body>
        <h1>Welcome to my web app!</h1>
        <p>Here are some names in the greeting servlet:</p>
        <ul>
            • <li><a href="/hello/Ada">Ada<a></li>
            • <li><a href="/hello/Grace">Grace<a></li>
            • <li><a href="/hello/Hal">Hal<a></li>
        </ul>
        <p><a href="/date.html">Here</a> is the current time servlet.</p>
    </body>
</html>
```

Enregistrez-le dans un fichier nommé index.html. Ensuite, ouvrez un navigateur /index.html et vous devriez voir ceci :



Ce nouveau fichier index.html est une page statique qui renvoie aux URL des servlets. Cette approche consistant à mélanger du contenu statique et dynamique est utile lorsque vous avez des sites qui devraient toujours être les mêmes (comme une page d'accueil ou une page à propos), mais d'autres pages qui devraient changer (comme des profils d'utilisateurs ou des données extraites d'une base de données). Il est également utile pour stocker des éléments tels que des fichiers CSS et des images, que vous pouvez utiliser dans votre HTML dynamique.

8. Sortie HTML

Dans les exemples ci-dessus, les servlets n'ont pas généré une page HTML complète. Je l'ai fait principalement pour les raccourcir, mais vous pouvez générer une page HTML complète à partir de votre servlet. Cela pourrait ressembler à ceci :

```
import java.io.PrintWriter;
import java.io.IOException;
import java.util.Date;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/date.html")
public class MyServlet extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
    IOException {
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>My Web App</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>My Web App</h1>");
        out.println("<p>The current time is: " + new Date().toString() + "</p>");
        out.println("</body>");
```

```
                out.println("</html>");
            }
        }
```

Ce code est conçu comme un exemple de sortie d'une page HTML complète pour le client. C'est assez difficile à manier, vous apprendrez donc de meilleures façons de le faire dans le prochain didacticiel.