Jakarta Server Pages (JSP)[1]

Vous savez maintenant comment ajouter des **servlets** à votre application Web. Une servlet ressemble à ceci :

```
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("/my-servlet")
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 utilise la `out.println()`fonction pour générer du HTML une ligne à la fois. En d'autres termes, il s'agit de HTML à l'intérieur de Java.

Il y a quelques problèmes avec le HTML dans Java :

- C'est difficile à éditer. Même ce petit bout de HTML est ennuyeux à travailler.

---

[1]
- *happycoding.io*
- *tutorialspoint.com/restful*
- *https://restfulapi.net*
- *oracle-base.com/articles/misc/*

- Il est difficile de formater avec une indentation ou une coloration syntaxique appropriée.
- C'est difficile à déboguer : comment trouver une faute de frappe au milieu d'un tas de valeurs de chaîne ?

En d'autres termes, c'est ennuyeux de travailler de cette façon.

Ce didacticiel présente JSP, ou Jakarta Server Pages , qui ressemblent davantage à Java dans HTML.

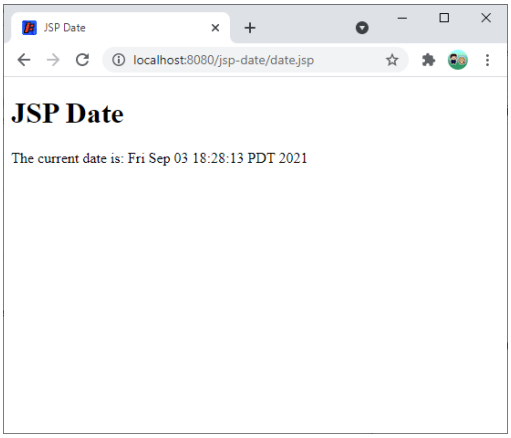
1. Fichiers JSP

Au lieu d'écrire du code Java qui génère du HTML, JSP vous permet d'écrire du code HTML qui contient du code Java. Voici un fichier JSP qui génère le même contenu dynamique (la date actuelle) que la servlet ci-dessus :

```
<%@ page import="java.util.Date" %>
<!DOCTYPE html>
<html>
    <head>
        <title>JSP Date</title>
    </head>
    <body>
        <h1>JSP Date</h1>
        <p>The current date is: <%= new Date() %></p>
    </body>
</html>
```

Enregistrez-le dans un fichier nommé date.jsp dans le répertoire de votre application Web à côté de votre fichier index.html.

Démarrez votre serveur. Ouvrez maintenant un navigateur sur date.jsp, et vous devriez voir ceci :



Lorsque vous visitez le fichier date.jsp, le serveur Jetty le compile automatiquement dans une classe de servlet, qui fonctionne alors comme n'importe quel autre servlet. Notez que cela nécessite un serveur Java ! Vous ne pouvez donc pas télécharger un fichier .jsp sur un hébergeur de fichiers. Il doit s'agir d'un serveur Java.

Mais maintenant, au lieu d'écrire du code Java contenant du code HTML, vous pouvez écrire du code HTML contenant du code Java. Cela rend beaucoup plus facile d'écrire une page Web.

Il existe plusieurs façons d'inclure du code Java dans une page JSP. Le reste de ce tutoriel les présente.

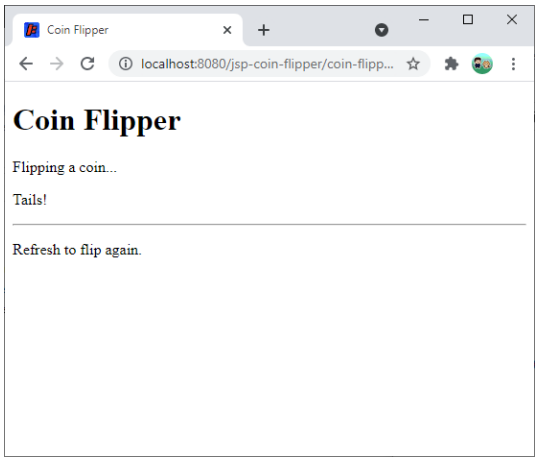
2. Scriptlets

Un scriptlet est un code Java qui **fait quelque chose** et qui est placé entre <%les %>balises d' ouverture et de fermeture de scriptlet. Voici un exemple :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Coin Flipper</title>
  </head>
  <body>
    <h1>Coin Flipper</h1>
    <p>Flipping a coin...</p>
    <% if(Math.random() < .5){ %>
      <p>Heads!</p>
    <% } else{ %>
      <p>Tails!</p>
    <% } %>
    <hr>
```

```
    <p>Refresh to flip again.</p>
  </body>
</html>
```

Ce code utilise une instruction `if` et la fonction `Math.random()` pour afficher `Heads!` ou `Tails!` au hasard.



Remarquez quelques éléments dans le code ci-dessus :

- Vous pouvez utiliser du code Java normal, comme des instructions fonction et des boucles.
- Le flux du code Java détermine la sortie HTML. Par exemple, ce code s'imprime uniquement `<p>Heads!</p>` ou en fonction `<p>Tails!</p>` du résultat de l'instruction `if`.
- Les scriptlets peuvent contenir plusieurs lignes de code.
- Vous pouvez diviser le code Java en plusieurs scriptlets. Par exemple, un scriptlet peut contenir l'ouverture de l'instruction `if` et un autre scriptlet contient la parenthèse fermante.

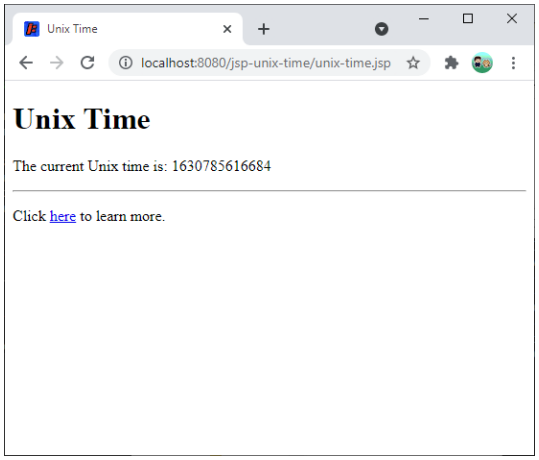
3. **Expressions**

Une expression est un code Java **évalué à une valeur** et placé entre `<%=` et `%>` balises d'expression d'ouverture et de fermeture . L'exemple `date.jsp` ci-dessus a utilisé une expression pour afficher la date actuelle. Voici un autre exemple :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Unix Time</title>
  </head>
  <body>
    <h1>Unix Time</h1>
    <p>The current Unix time is: <%= System.currentTimeMillis() %></p>
```

```
  <hr>
 <p>Click <a href="https://en.wikipedia.org/wiki/Unix_time">here</a> to learn more.</p>
 </body>
</html>
```

Ce code utilise une expression pour insérer la valeur renvoyée par la `System.currentTimeMillis()`fonction dans le HTML de la page.



4. Mélanger des scriptlets et des expressions

Vous pouvez utiliser à la fois des scriptlets (code qui fait quelque chose) et des expressions (code qui évalue une valeur) dans le même fichier JSP. Voici un exemple :

- `<% String message = "hello!"; %>`
- `<p>Here's some HTML</p>`
- `<p>Message: <%= message %></p>`

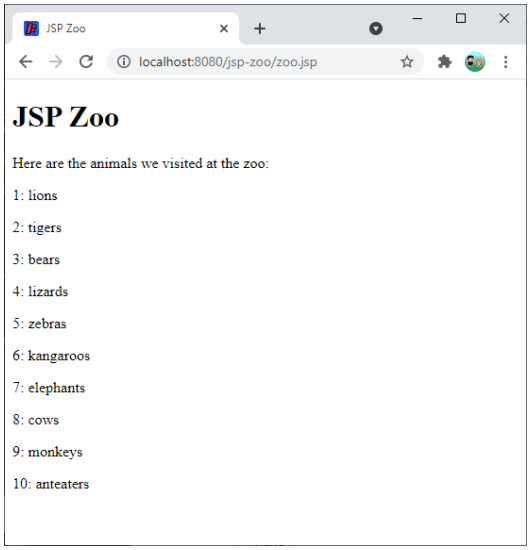
Ce code contient un scriptlet qui crée une variable `message`. Ensuite, il utilise une expression pour insérer la valeur de cette variable dans le code HTML.

Voici un exemple plus compliqué :

```
<% String[] animals = {
 "lions", "tigers", "bears", "lizards", "zebras",
 "kangaroos", "elephants", "cows", "monkeys", "anteaters"
 }; %>
<!DOCTYPE html>
<html>
 <head>
```

```
  <title>JSP Zoo</title>
 </head>
 <body>
  <h1>JSP Zoo</h1>
  <p>Here are the animals we visited at the zoo:</p>
  <% for(int i = 0; i < animals.length; i++){ %>
   <p><%= i+1 %>: <%= animals[i] %></p>
  <% } %>
 </body>
</html>
```

Ce code utilise un scriptlet pour définir un tableau nommé animals. Ensuite, il utilise un autre scriptlet pour parcourir ce tableau à l'aide d'une boucle for. À l'intérieur de la boucle for, il utilise une expression pour imprimer l'index et l'animal à cet index. Enfin, il utilise un autre scriptlet pour fermer la boucle.



5. Directives

Jusqu'à présent, vous avez appris que les scriptlets contiennent du code qui fait quelque chose et que les expressions contiennent du code qui évalue une valeur.

Les directives sont placées entre <%@ et %>balises d' ouverture et de fermeture, et elles contiennent du code qui indique à la page elle-même ce qu'elle doit faire.

Les directives incluent des éléments tels que des instructions Java import, comme dans la première ligne de ce code :

```
<%@ page import="java.util.Date" %>
```

```
<!DOCTYPE html>
<html>
	<head>
		<title>JSP Date</title>
	</head>
	<body>
		<h1>JSP Date</h1>
		<p>The current date is: <%= new Date() %></p>
	</body>
</html>
```

Une directive pratique est la directive include, qui vous permet d'inclure le contenu d'autres fichiers. Par exemple, vous pouvez l'utiliser pour créer une barre de navigation dans un fichier, puis inclure ce fichier dans toutes vos pages JSP.

Enregistrez ce contenu dans un fichier nommé header.html:

```
<div style="border: thin solid black; padding:5px;">
  <a href="index.jsp">Home</a>
  <a href="about.jsp">About</a>
  <a href="pictures.jsp">Pictures</a>
</div>
```

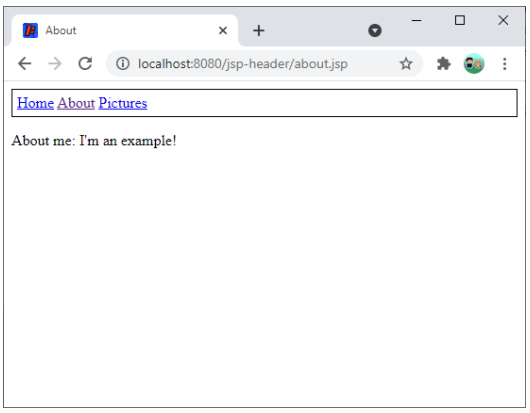
Maintenant que vous avez ce fichier, vous pouvez utiliser la directive include pour l'inclure dans vos pages JSP :

```
<!DOCTYPE html>
<html>
  <head>
    <title>About</title>
  </head>
  <body>

    <%@ include file = "header.html" %>

    <p>About me: I'm an example!</p>
  </body>
</html>
```

La directive include à l'intérieur de la <body> balise copie le contenu depuis header.html dans le contenu du fichier JSP, vous voyez donc l'en-tête dans votre page :



Cela facilite le partage de contenu entre plusieurs pages. Si vous aviez 100 pages qui incluaient l'en-tête, vous n'auriez besoin de modifier qu'un seul fichier pour changer l'en-tête sur chaque page.

6. Les erreurs

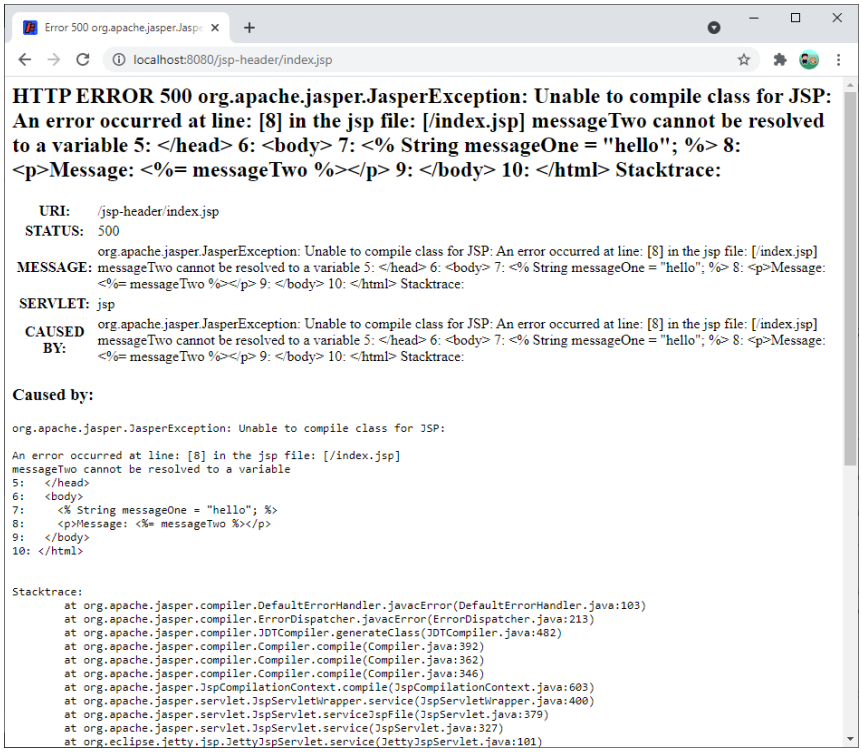
N'oubliez pas que les fichiers JSP sont automatiquement compilés dans des classes de servlet. Cela signifie que vous devez faire attention à la sortie, afin que vous sachiez quand des erreurs se produisent.

Prenons cet exemple JSP :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Error Example</title>
  </head>
  <body>
    <% String messageOne = "hello"; %>
    <p>Message: <%= messageTwo %></p>
  </body>
</html>
```

Ce code crée une messageOnevariable, mais essaie ensuite de lire une variable messageTwo. La variable messageTwo n'existe pas, donc ce code ne sera pas compilé.

Si vous essayez de visiter cette page, vous verrez une erreur :



C'est écrasant au début, mais vous pouvez l'utiliser pour comprendre quelle est l'erreur.

org.apache.jasper.JasperException: Unable to compile class for JSP:

An error occurred at line: [8] in the jsp file: [/index.jsp]
messageTwo cannot be resolved to a variable
5: </head>
6: <body>
7: <% String messageOne = "hello"; %>
8: Message: <%= messageTwo %>
9: </body>
10: </html>

Cela vous indique que l'erreur est à la ligne 8 et qu'il ne peut pas trouver la variable messageTwo.

Pour le meilleur ou pour le pire, rencontrer de telles erreurs fait naturellement partie de la programmation !

7. Mélange de servlets et de fichiers JSP

Vous savez maintenant utiliser les **servlets** . Une servlet ressemble à ceci :

```
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("/my-servlet")
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>");
    }
}
```

Et vous savez utiliser les fichiers **JSP** . Un fichier JSP ressemble à ceci :

```
<%@ page import="java.util.Date" %>
<!DOCTYPE html>
<html>
    <head>
        <title>JSP Date</title>
    </head>
    <body>
        <h1>JSP Date</h1>
        <p>The current date is: <%= new Date() %></p>
```

```
        </body>
</html>
```

Il y a des avantages et des inconvénients à chaque approche. Les servlets facilitent l'utilisation du code Java, mais il est ennuyeux d'y programmer du HTML. D'un autre côté, les fichiers JSP facilitent l'écriture HTML, mais il peut être ennuyeux d'inclure une logique Java compliquée dans un fichier JSP.

Vous pouvez utiliser un mélange de servlets et de fichiers JSP pour tirer le meilleur parti des deux mondes : vous pouvez utiliser des servlets pour exécuter votre code Java et vous pouvez utiliser des fichiers JSP pour afficher une page à l'aide des paramètres générés à partir du servlet.

Par exemple, reprenons l'exemple ci-dessus pour utiliser une servlet qui formate l'heure actuelle et la transmet en paramètre à un fichier JSP pour le rendu.

Voici à quoi ressemblerait la servlet :

```
package io.happycoding.servlets;

import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.ServletException;

@WebServlet("/date")
public class DateServlet extends HttpServlet {

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

    SimpleDateFormat dateFormat =
        new SimpleDateFormat("hh:mm aa 'on' EEEE, MMMM dd, yyyy");
    Date now = new Date();
    String formattedDate = dateFormat.format(now);

    request.setAttribute("date", formattedDate);
    request.getRequestDispatcher("/date-view.jsp").forward(request,response);
```

```
    }
}
```

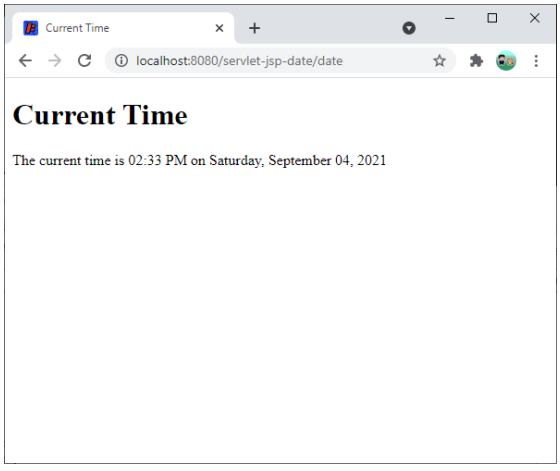
Ce code utilise la classe `SimpleDateFormat` pour formater l'heure actuelle. Il utilise ensuite la fonction `request.setAttribute()` pour inclure la date formatée dans la demande, puis il utilise la fonction `request.getRequestDispatcher("/date-view.jsp").forward(request, response)` pour envoyer la demande à un fichier JSP.

Ensuite, le fichier `date-view.jsp` gère la requête :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Current Time</title>
  </head>
  <body>
    <h1>Current Time</h1>
    <p>The current time is <%= request.getAttribute("date") %></p>
  </body>
</html>
```

Notez l'expression `<%= request.getAttribute("date") %>`, qui obtient l'attribut `date` ajouté par la servlet.

Cela vous permet de conserver toute votre logique Java dans vos classes de servlet et vos fichiers JSP génèrent le contenu sans vous soucier du code logique compliqué.



8. Langage d'expression

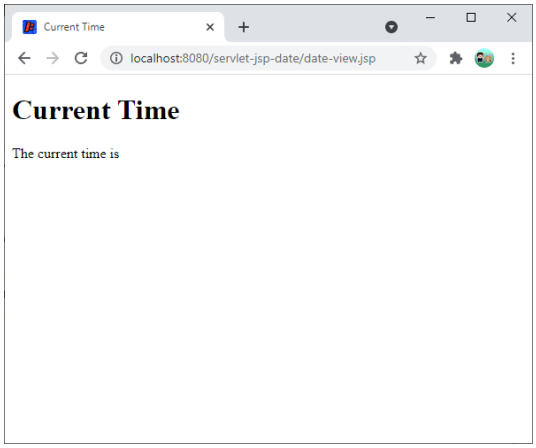
L'exemple ci-dessus utilise le `request.getAttribute("date")` dans JSP pour rendre plus évident ce qui se passe, mais vous pouvez utiliser **un langage d'expression** pour raccourcir cela.

Le langage d'expression (ou **EL**) fournit une syntaxe abrégée pour faire des choses comme accéder aux attributs. Avec EL, vous pouvez utiliser `${date}` à la place de `<%= request.getAttribute("date") %>`:

```
<!DOCTYPE html>
<html>
  <head>
  <title>Current Time</title>
  </head>
  <body>
    <h1>Current Time</h1>
    <p>The current time is ${date}</p>
  </body>
</html>
```

9. Cacher les fichiers JSP

L'exemple ci-dessus présente un problème potentiel : le fichier `date-view.jsp` est accessible au public. Cela signifie que les utilisateurs peuvent visiter `date-view.jsp` directement, ce qui affiche une page comme celle-ci :



Notez que l'heure actuelle est vide. Étant donné que l'utilisateur a navigué directement, la servlet `date-view.jsp` n'a pas été déclenchée, elle n'a donc pas ajouté l'heure actuelle aux attributs de la requête.

Vous pouvez empêcher les utilisateurs de voir des pages boguées comme celle-ci en masquant vos fichiers JSP.

Pour masquer un fichier JSP, créez un répertoire WEB-INF et déplacez vos fichiers JSP dans ce répertoire.

- Si vous créez manuellement votre répertoire d'applications Web, vous disposez déjà d'un répertoire WEB-INF qui contient votre répertoire classes. Votre répertoire d'applications Web ressemblerait à ceci :
    - MyWebApp/
        - index.html
        - WEB-INF/
            - date-view.jsp
            - classes/io/happycoding/servlets/
                - DateServlet.class

Maintenant que votre fichier date-view.jsp est caché dans le répertoire WEB-INF, modifiez la classe DateServlet pour transmettre la requête au nouvel emplacement de date-view.jsp :

```
package io.happycoding.servlets;


    import java.io.IOException;
    import java.text.SimpleDateFormat;
    import java.util.Date;
    import jakarta.servlet.annotation.WebServlet;
    import jakarta.servlet.http.HttpServlet;
    import jakarta.servlet.http.HttpServletRequest;
    import jakarta.servlet.http.HttpServletResponse;
    import jakarta.servlet.ServletException;

    @WebServlet("/date")
    public class DateServlet extends HttpServlet {

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

        SimpleDateFormat dateFormat =
            new SimpleDateFormat("hh:mm aa 'on' EEEE, MMMM dd, yyyy");
        Date now = new Date();
        String formattedDate = dateFormat.format(now);

        request.setAttribute("date", formattedDate);
```

```
        request.getRequestDispatcher("/WEB-INF/date-view.jsp").forward(request,response);
    }
}
```

Ce code transmet maintenant la demande à `/WEB-INF/jsp/date-view.jsp`. Votre code est autorisé à accéder au répertoire `WEB-INF`, mais les utilisateurs ne peuvent pas voir ce qu'il contient. Cela vous permet de garder les choses principalement cachées aux utilisateurs.

Essayez d'accéder `date-view.jsp` directement au fichier. Vous devriez plutôt voir une erreur 404, car le fichier n'est plus accessible directement.

10. MVC

Cette approche consistant à utiliser des servlets pour contenir votre logique et des fichiers JSP pour contenir votre code de rendu est une grande partie du modèle de conception Model-View-Controller , ou MVC.

MVC est l'idée de séparer vos données de votre logique de votre rendu. Il vous permet de vous concentrer sur une chose à la fois et facilite le débogage et la modification d'une seule pièce. Vous pouvez utiliser cette idée dans n'importe quel type de programmation, mais lors du développement de code serveur, cela fonctionne généralement comme ceci :

- Votre **modèle** est constitué des structures de données qui contiennent les données de votre site. Cela peut être des choses comme une base de données ou des objets `ArrayList` qui contiennent des données.
- Votre **contrôleur** est votre classe de servlet. Ceux-ci lisent à partir de votre modèle et exécutent la logique.
- Votre **vue** est vos fichiers JSP. Ceux-ci prennent les paramètres de votre contrôleur pour rendre une page.

Vous n'avez pas à vous soucier d'obtenir tout ce qui est exactement correct, et des concepts comme MVC sont plus une façon de penser l'organisation qu'un ensemble de règles strictes. Mais garder vos données, votre logique et votre rendu séparés vous facilitera certainement la vie. Et cette approche consistant à diviser vos servlets et vos fichiers JSP vous aidera.

Fichiers statiques

Vous pouvez toujours utiliser des fichiers statiques avec vos servlets et fichiers JSP. Vous pouvez avoir des fichiers HTML statiques liés à des servlets et des fichiers JSP liés à des fichiers HTML statiques, et tout le reste.

N'oubliez pas : le résultat final qui est envoyé à votre navigateur est du simple HTML !