Servlets et JSP : la JSTL

1 Utiliser le langage d'expression (EL)

Depuis la version 2.0 des JSP, il est possible de placer à n'importe quel endroit d'une page JSP des expressions qui sont évaluées et remplacées par le résultat de leur évaluation. La syntaxe est la suivante :

```
${ expression }
```

Voila quelques exemples d'utilisation :

- `${bean.property}` accès à une propriété d'un bean,
- `${bean.property1.property2}` suivre un chemin pour avoir accès à une propriété d'un bean,
- `${list[index]}` accès à un élément indicé (`Array` , `Map` , `List`),

Travail à faire :

- Prenez un peu de temps pour lire cette présentation du langage d'expressions [1].
- Vous pouvez maintenant modifier votre TP précédent en utilisant les *expressions* à la place des *scriptlets* pour initialiser les champs de votre formulaire et/ou visualiser les propriétés du bean personne.
- Préparez le javabean ci-dessous :

```java
package myapp;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Student {

    private String name;
    private Student bestFriend;
    private Integer age;

}
```

- Préparez la servlet ci-dessous. Cette dernière prépare des données et appelle la page `students.jsp` :

1. http://adiguba.developpez.com/tutoriels/j2ee/jsp/el/

```java
package myapp;

import java.io.IOException;
import java.util.Arrays;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/students")
public class Students extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public void doGet(
        HttpServletRequest request,
        HttpServletResponse response
    ) throws ServletException, IOException {
        // les étudiants
        Student john = new Student("John", null, 23);
        Student anne = new Student("Anne", john, 25);
        Student amine = new Student("Amine", anne, 27);
        // la liste des étudiants
        List<Student> students = Arrays.asList(john, anne, amine);
        // les étudiants par leur nom
        Map<String, Student> names = new HashMap<>();
        students.forEach((Student s) -> {
            names.put(s.getName(), s);
            request.setAttribute(s.getName(), s);
        });

        request.setAttribute("students", students);
        request.setAttribute("names", names);
        request.getSession().setAttribute("anne", anne);
        // le jour
        request.setAttribute("today", new Date());
        request.getSession().setAttribute("today", "Hier");
        request.getSession().getServletContext().setAttribute("today", "Avant-hier");
        // appel de la page JSP
        request.getRequestDispatcher("/students.jsp").forward(request, response);
    }

}
```

- Terminez par la page JSP et testez la servlet :

students.jsp

```jsp
<html>
<body>
  <p>Age de Anne est ${anne.age}</p>
  <p>Aujourd'hui nous sommes le ${today}</p>
</body>
</html>
```

- Préparez, dans la page JSP, un paragraphe pour afficher le nom du meilleur amie de Anne. Faites de même pour John. Vous observerez la gestion assez souple des null.

- Préparez un paragraphe pour afficher le troisième étudiant (en utilisant la liste `students`).
- Il est possible d'appeler une méthode avec la forme `nom_de_méthode()`. Affichez ainsi le nombre d'étudiants (méthode `size()`).
- Affichez ensuite la date (variable `today`). Faites de même pour la version stockée en session en utilisant la variable implicite `sessionScope`. Continuez avec la version stockée dans la zone application (variable implicite `applicationScope`).
- Préparez, dans la servlet, une nouvelle donnée qui regroupe les caractères spéciaux de HTML (`<>&'"`). Affichez cette donnée avec une EL. Les caractères spéciaux ne sont pas protégés et nous pouvons donc avoir **une injection HTML/javascript**.

2 Java Standard Tag Library

2.1 Récupérer et installer la JSTL 1.2

Documentations :

- La page sur la JSTL chez Java Soft [2]
- Les transparents sur les librairies de balises [3]
- Un cours sur la JSTL [4]

Travail : Ajoutez la dépendance ci-dessous dans le fichier `pom.xml` :

```
<!-- pour utiliser la JSTL -->
<dependency>
    <groupId>jakarta.servlet.jsp.jstl</groupId>
    <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
    <version>3.0.0</version>
</dependency>
<dependency>
    <groupId>org.glassfish.web</groupId>
    <artifactId>jakarta.servlet.jsp.jstl</artifactId>
    <version>3.0.1</version>
</dependency>
```

Remarque : Je vous encourage fortement à lire (après le TP) le cours sur la JSTL (lien ci-dessus). Il explique bien le fonctionnement de ses balises et donne de nombreux exemples.

2.2 Utiliser la JSTL

Pour utiliser la JSTL 1.2.x dans une application basée sur un conteneur WEB respectant le standard JSP 2.0, il suffit de copier dans le répertoire `WEB-INF/lib` de votre application WEB les fichiers `*.jar` qui se trouvent dans la distribution de la JSTL. **Ce travail est déjà réalisé par Maven**.

Il est notamment inutile de copier les fichiers `.tld`. Ceux-ci se trouvent dans les `.jar` et sont pris en compte automatiquement. Une fois le contexte de votre application relancé, vous pouvez essayer la page suivante :

2. http://www.oracle.com/technetwork/java/index-jsp-138231.html
3. jsp.html#taglib
4. http://adiguba.developpez.com/tutoriels/j2ee/jsp/jstl/

```
<html>
<body>

<!-- controle, iterations, tests, variables -->
<%@ taglib prefix="c" uri="jakarta.tags.core" %>
<!-- traitement XML -->
<%@ taglib prefix="x" uri="jakarta.tags.xml" %>
<!-- formattage des donnees -->
<%@ taglib prefix="fmt" uri="jakarta.tags.fmt" %>
<!-- SQL/JDBC -->
<%@ taglib prefix="sql" uri="jakarta.tags.sql" %>

<!-- Un exemple de test -->
<c:if test="${param.size() > 0}">
    <p>Il y a des paramètres.</p>
</c:if>

<!-- Un exemple de boucle -->
<p>Les parametres:</p>
<ul>
<c:forEach var="aParam" items="${param}">
  <li>un parametre :
    <c:out value="${aParam.key}"/> = <c:out value="${aParam.value}"/>
  </li>
</c:forEach>
</ul>

<!-- Une étude de cas -->
<c:choose>
  <c:when test="${param['question'] == 'oui'}">
     <p>OUI</p>
  </c:when>
  <c:otherwise>
     <p>NON</p>
  </c:otherwise>
</c:choose>

</body>
</html>
```

Travail à faire :

- Modifiez vos pages JSP afin d'utiliser la balise `c:out` (voir exemple ci-dessus). Vous ne devriez plus avoir d'injection de code HTML.
- Utilisez dans vos pages JSP la balise `c:url` pour construire les URL des liens vers les servlets ou les autres pages JSP (exemple ci-dessous). Vérifiez le code généré par cette balise.

```
<%@ taglib prefix="c" uri="jakarta.tags.core" %>

...

<c:url var="lister" value="/lister.jsp" />

<p><a href="${lister}">Lister</a></p>
```

- Construisez une boucle (avec `c:forEach`) pour afficher un tableau des étudiants (nom, age et meilleur ami). Si l'age est supérieur à 24, l'afficher en gras (avec `c:if` ou `c:choose`). Si l'étudiant n'a pas de meilleur ami, affichez la chaîne `personne` (avec `c:out` et l'attribut `default`).

3 Les filtres

Créez dans votre application le filtre ci-dessous (en adaptant l'URL des adresses à traiter) :

```
package myapp;

import java.io.IOException;
import jakarta.servlet.DispatcherType;
import jakarta.servlet.Filter;
import jakarta.servlet.FilterChain;
import jakarta.servlet.FilterConfig;
import jakarta.servlet.ServletException;
import jakarta.servlet.ServletRequest;
import jakarta.servlet.ServletResponse;
import jakarta.servlet.annotation.WebFilter;
import jakarta.servlet.http.HttpServletRequest;

@WebFilter(
    dispatcherTypes = {
        DispatcherType.REQUEST, DispatcherType.FORWARD,
        DispatcherType.INCLUDE, DispatcherType.ERROR
    },
    urlPatterns = { "*" }
)
public class SimpleFilter implements Filter {

    public void init(FilterConfig fConfig) throws ServletException {
    }

    public void destroy() {
    }

    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException
    {
        System.err.printf("Before␣%s\n", request);
        if (request instanceof HttpServletRequest) {
            HttpServletRequest hr = (HttpServletRequest) request;
            System.err.printf("Before␣Http␣Request␣%s\n", hr);
        }
        chain.doFilter(request, response);
        System.err.printf("After␣%s\n", request);
    }

}
```

et vérifier son fonctionnement. Il permet d'insérer des traitements automatisés avant et après chaque requête et donc d'appliquer des **politiques de sécurité**.

Travail à faire : Avec le filtre, interdisez les requêtes qui ne proviennent pas de l'adresse 127.0.0.1 (utilisez `request.getRemoteAddr()`). L'interception consiste à ne pas appeler la méthode `doFilter`.

4 Les listeners

Ajoutez à votre application, la classe d'écoute :

```java
package myapp;

import jakarta.servlet.annotation.WebListener;
import jakarta.servlet.http.HttpSessionAttributeListener;
import jakarta.servlet.http.HttpSessionBindingEvent;

@WebListener
public class SimpleListener implements HttpSessionAttributeListener {

    @Override
    public void attributeAdded(HttpSessionBindingEvent event) {
    }

    @Override
    public void attributeRemoved(HttpSessionBindingEvent event) {
    }

    @Override
    public void attributeReplaced(HttpSessionBindingEvent event) {
    }

}
```

et explorez les possibilités de l'argument passé à chaque méthode. Vous pouvez également vous intéresser aux interfaces

- `ServletContextListener`
- `ServletContextAttributeListener`
- `ServletRequestListener`
- `ServletRequestAttributeListener`
- `HttpSessionListener` création/suppression de sessions