Mise en oeuvre des Servlets et des JSP

1 Le conteneur WEB Tomcat d'Apache (5m)

- Commencez par récupérer le conteneur WEB Tomcat [1] (ou ici [2]) (version **9.0.73** non RPM, package **core**) et installez-le (voir commandes ci-dessous) :

```
cd "$HOME"
wget http://tinyurl.com/jlmassat/ens/jee/ress/apache-tomcat-9.0.73.zip
unzip apache-tomcat-9.0.73.zip
chmod u+x apache-tomcat-9.0.73/bin/*.sh
export TOMCAT=$HOME/apache-tomcat-9.0.73
```

- **Travail à faire plus tard** :
  - ▷ Lancez le serveur avec le script `$TOMCAT/bin/startup.sh`.
  - ▷ Testez ensuite les exemples présents en vous connectant à l'adresse `http://localhost:8080`. **Attention** : commencez par les exemples de servlets puis les exemples en JSP 1.2 et laissez les exemples JSP 2.0 pour plus tard.
  - ▷ **N'oubliez pas**, à la fin de cet exercice, de stopper Tomcat (commande `$TOMCAT/bin/shutdown.sh`).

2 Eclipse JEE et les applications WEB (20m)

Nous allons utiliser le plugin `WTP` [3] qui est intégré par défaut dans Eclipse pour JEE. Pour ce faire, suivez ces étapes **les unes après les autres** :

1. Lancer la version JEE de Eclipse [4].
2. Dans le menu « **Windows/Preferences** » choissisez l'onglet « **Server / Runtime Environments** » et ajoutez un nouveau **Runtime** de type **Apache Tomcat 9.0**.
3. Repérez la vue **Server** (menu « **Windows / Show view** ») et créez un nouveau serveur (avec le menu contextuel dans la vue **Server**) basé sur le **Runtime Tomcat 9** précédemment créé.
4. Créez un nouveau projet de type « **Web / Dynamic Web project** » :
   - **Écran 1** : Fixez le nom `myapp`, choisissez le **Target Runtime** Apache Tomcat 9 et la version 4.0 du module.
   - **Écran 2** : Configuration du répertoire source (rien à faire).
   - **Écran 3** : Configuration du contexte et du répertoire contenant les ressources WEB (rien à faire). **À ce stade, il faut** choisir l'option « Generate web.xml » afin qu'eclipse prépare automatiquement le fichier de configuration.
5. Convertissez votre projet à Maven : **Sélectionnez votre projet** / **Bouton-droit** / **Configure** / **Convert to Maven Project**. Vous devez à cette étape donner une numéro de version à votre projet. Laissez les valeurs par défaut.
6. Ajoutez les dépendances ci-dessous dans le fichier `pom.xml` :

1. http://tomcat.apache.org/
2. ref:ress
3. http ://www.eclipse.org/webtools/
4. http://www.eclipse.org/

```xml
<!-- Pour utiliser Lombok -->
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.22</version>
    <scope>provided</scope>
</dependency>
<!-- Pour valider des données -->
<dependency>
    <groupId>commons-validator</groupId>
    <artifactId>commons-validator</artifactId>
    <version>1.7</version>
</dependency>
```

7. À ce stade, votre projet doit ressembler à ceci :

```
myapp
  | Deployment Descriptor     version agréable de web.xml
  | JAX-WS Web Services       pas nécessaire dans l'immédiat
  | JRE System...             la JRE
  | src/main/java             les codes Java
  | Server Runtime (Tomcat...) les librairies de Tomcat
  + Deployed Resources
  |   + webapp                votre application WEB
  |   |   + META-INF          le manifest
  |   |   + WEB-INF           configuration de votre app.
  |   |   |   | lib/          les librairies de votre app
  |   |   |   | web.xml       configuration de votre app
  |   + web-resources         les ressources
  | build/                    zone de travail
  | src/                      une autre vue des sources
  | target/                   zone de travail maven
  | pom.xml                   fiche de configuration maven
```

8. Créez dans le répertoire `webapp` une page JSP et nommez-la `index.jsp`. Elle doit ressembler à ceci

Fichier `webapp/index.jsp`

```jsp
<%@ page language="java"
    contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Title</title>
</head>
<body>
  <p>Hello.</p>
</body>
</html>
```

9. Sélectionnez votre projet et exécutez-le sur le serveur Tomcat (menu « **Run as... / Run on server** »). Faites en sorte d'associer définitivement votre projet et le serveur Tomcat (une petite case à cocher avant le lancement).

10. À ce stade, vous devez pouvoir accéder à votre application via l'URL (`http://localhost:8080/myapp/`). Vous pouvez maintenant ajouter des pages HTML, JSP et/ou des sources Java dans votre projet.

3 Mon premier bean (20m)

- Modifiez la page précédente pour qu'elle affiche la date du jour et l'heure à chaque exécution (créez pour cela une instance de la classe `java.util.Date` et affichez la).

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

<%!
Date now = new Date();
%>

<p>Aujourd'hui : <%= now %></p>
```

  Quel comportement anormal observez-vous ? Essayez cette nouvelle version :

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

<p>Aujourd'hui : <%= new Date() %></p>
```

- Faites tourner les exemples présentés en cours[5] d'affichage et de manipulation d'un produit.
- Faites varier la portée (le `scope`) du `<jsp:useBean>` et observez les différences dans les trois cas[6] (`request`, `session`, `application`).
  - ▷ **Conseil 1** : Vous pouvez placer du code JSP à l'intérieur de l'action `<jsp:useBean>`. Ce code est exécuté lorsque le bean est créé.
  - ▷ **Conseil 2** : Pour bien manipuler les *beans* de portée `session`, utilisez plusieurs navigateurs (firefox ou chrome) ou une fenêtre de navigation privée.
  - ▷ **Conseil 3** : Faites en sorte que votre bean implante l'interface `HttpSessionBindingListener`[7] et mettez en place des traces pour suivre la vie de ce bean. Dans un deuxième temps, diminuez la durée de vie des sessions (ajoutez le code ci-dessous dans le fichier `web.xml`) afin d'observer la suppression automatique des beans (**attention** : c'est loin d'être systématique).

```
...
<session-config>
   <session-timeout>1</session-timeout><!-- une minute -->
</session-config>
...
```

4 Une petite application (1h40)

4.1 Étape 1 : présenter

- Créez un « bean » `Person` représentant une personne :

5. jsp.html#javabean
6. servlet.html#scope
7. servlet.html#HttpSessionBindingListener

```java
package myapp;

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

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Person {

    private Integer id;
    private String name;
    private String mail;

}
```

- Créez une page JSP `person.jsp` qui présente (dans un tableau HTML) le contenu d'une instance de `Person` accessible en session (faites le lien avec `<jsp:useBean>`).
- **Conseil** : Testez cette page JSP. Comment éviter (simplement) les `null` ?
- Créez une servlet `PersonServlet` que ne réalise aucune action (pour l'instant) et qui est associée à l'URL `/myapp/edition` (code ci-dessous).

```java
package myapp;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Une servlet pour les actions sur les personnes.
 */
@WebServlet(//
        description = "Les actions sur les personnes", //
        urlPatterns = { "/edition" })
public class PersonServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * Requêtes GET
     */
    protected void doGet(HttpServletRequest request, //
            HttpServletResponse response) //
            throws ServletException, IOException {
        response.getWriter()//
                .append("Served at: ")//
                .append(request.getContextPath());
    }

    /**
     * Requêtes POST
     */
    protected void doPost(HttpServletRequest request, //
            HttpServletResponse response) //
            throws ServletException, IOException {
        doGet(request, response);
    }

}
```

- Créez une page JSP `edition.jsp` afin de produire un formulaire HTML d'édition des caractéristiques d'une personne placée en session. La soumission de ce formulaire va appeler la servlet `PersonServlet`. **Conseil** : aidez-vous du cours[8].

4.2 Étape 2 : traiter

Modifiez la servlet afin qu'elle réalise les actions suivantes (dans la méthode `doPost`)

- Créer une instance de `Person` ou la récupérer à partir de la session si elle existe déjà (prévoir une trace sur `stderr` de cette création). Aidez-vous de la méthode ci-dessous.

8. servlet.html#servlet-form

```java
/**
 * Récupérer/créer une donnée bien typée en session
 */
private Person getOrCreateSessionPerson(HttpServletRequest request) {
    var session = request.getSession();
    var object = session.getAttribute("person");
    if (object instanceof Person) {
        return (Person) object;
    }
    var p = new Person();
    session.setAttribute("person", p);
    return p;
}
```

- Affecter cette instance avec les paramètres de la requête HTTP (id, nom, mail).
- Appeler la page JSP `person.jsp` en utilisant le code ci-dessous :

```java
// Appeler une page JSP depuis une servlet
request.getRequestDispatcher(pageJsp).forward(request, response);
```

À faire : à ce stade, vous pouvez renseigner le formulaire et valider, votre application doit afficher les données.

4.3 Étape 3 : ajouter une couche métier

Modifiez votre projet afin d'ajouter une classe métier orientée vers le traitement des personnes :

```java
package myapp;

import java.util.Collection;
import java.util.Collections;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map;

public class PersonManager {

    final private Map<Integer, Person> persons;

    public PersonManager() {
        persons = Collections.synchronizedMap(new HashMap<>());
        save(new Person(100, "Paul", "paul@hello.fr"));
        save(new Person(200, "Laure", "laure@univ-amu.fr"));
    }

    private Person duplicate(Person p) {
        return new Person(p.getId(), p.getName(), p.getMail());
    }

    public Person find(Integer id) {
        var p = persons.get(id);
        return (p == null) ? null : duplicate(p);
    }

    public Collection<Person> findAll() {
        var result = new LinkedList<Person>();
        persons.values().forEach((Person p) -> {
            result.add(duplicate(p));
        });
        return result;
    }

    public void save(Person p) {
        var saved = duplicate(p);
        persons.put(saved.getId(), saved);
    }

    public boolean check(Person p) {
        throw new IllegalStateException("Not␣yet␣implemented");
    }

}
```

4.4 Étape 4 : valider

- Ajoutez à votre bean les propriétés `errorName` et `errorMail`. La méthode `check` va renseigner ces propriétés en cas d'erreur.
- Dans le Manager : Terminez la méthode `check` : le nom est obligatoire et l'email doit être valide (utilisez ensuite le code ci-dessous pour valider une adresse électronique)

```
EmailValidator.getInstance().isValid(mail)
```

- Dans la Servlet : Ajoutez une phase de validation des données. Faites en sorte, si les données ne sont pas valides, de revenir au formulaire en proposant les anciennes valeurs.
- Modifiez la page `edition.jsp` et la servlet de manière à faire apparaitre des messages d'erreur (en rouge)

à coté des champs fautifs.

4.5 Étape 5 : enregistrer et lister

- Dans la Servlet : Si les données sont valides, enregistrez l'instance indexée par le numéro de la personne.
- Créez la page JSP `lister.jsp` qui va lister les personnes stockées et prévoir, pour chacune, un lien vers la servlet de la forme

```
<a href="edition?numero=12345">Nom d'une personne</a>
```

  La servlet va utiliser la méthode HTTP (méthode `getMethod` [9]) pour savoir si elle doit lancer l'édition d'une personne identifiée par un numéro (méthode `GET`) ou valider et enregistrer une personne (méthode `POST`).

4.6 Étape 6 : ajouter

Ajoutez à votre page `lister.jsp` le lien ci-dessous afin de pouvoir ajouter une nouvelle personne :

```
<a href="edition">Ajouter une personne</a>
```

Important : prévoyez de placer en session une information sur le type d'édition en cours : création **ou** modification. En cas de modification, vous pourriez stocker l'identifiant de l'objet en cours d'édition.

4.7 Étape 7 : supprimer

Modifiez votre page `lister.jsp` et votre servlet pour ajouter et traiter le cas de la suppression :

```
<a href="supprimer?numero=123456">Supprimer cette personne</a>
```

Prévoyez que votre servlet traite également l'URL `/myapp/supprimer` en modifiant l'annotation :

```
@WebServlet(//
        description = "Les␣actions␣sur␣les␣personnes", //
        urlPatterns = { "/edition", "/supprimer" })
```

La servlet utilisera la méthode `request.getServletPath()` pour savoir sous quel nom elle a été appelée.

9. https://javaee.github.io/javaee-spec/javadocs/javax/servlet/http/HttpServletRequest.html#getMethod–