Spring Boot : formulaires et sessions

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

1. Formulaires
   - Récupérer les données d'un formulaire dans un objet
   - Valider les formulaires
2. Autres balises de formulaires Spring
3. Sessions

Spring Boot

Formulaire

- n'est pas un composant propre à **Spring**
- peut donc être construit intégralement avec **HTML**

Spring Boot

Formulaire

- n'est pas un composant propre à **Spring**
- peut donc être construit intégralement avec **HTML**

Apport de **Spring**

- Facilite la récupération de données envoyées par un formulaire dans un objet
- Simplifie la validation de formulaires

Spring Boot

Considérons la partie de `PersonneController` qui permet d'ajouter une personne

```
@Controller
public class PersonneController {

        @Autowired
        PersonneRepository personneRepository;

        @GetMapping("addPersonne")
        public String addPersonne() {
                return "addPersonne";
        }

        @PostMapping("addPersonne")
        public String addPersonne(@RequestParam String nom,
           @RequestParam String prenom, Model model) {
                model.addAttribute("nom", nom);
                model.addAttribute("prenom", prenom);
                Personne personne = new Personne(nom, prenom);
                personneRepository.save(personne);
                return "redirect:showPersonnes";
        }
}
```

Spring Boot

Constats

- Récupération des champs de formulaire un par un,
- Pas de contrôle de valeurs saisies par l'utilisateur et aucune gestion d'erreurs.

Spring Boot

Constats

- Récupération des champs de formulaire un par un,
- Pas de contrôle de valeurs saisies par l'utilisateur et aucune gestion d'erreurs.

Solutions

- Utiliser l'annotation `@ModelAttribute` pour récupérer les données envoyées par le formulaire dans un objet,
- Construire le formulaire en utilisant les balises **Spring**.

Spring Boot

Commençons par utiliser `@ModelAttribute` pour récupérer les données envoyées par le formulaire dans un objet

```
@Controller
public class PersonneController {

        @Autowired
        PersonneRepository personneRepository;

        @GetMapping("addPersonne")
        public String addPersonne(Model model) {
                model.addAttribute("personne", new Personne());
                return "addPersonne";
        }

        @PostMapping("addPersonne")
        public String addPersonne(@ModelAttribute("personne") Personne
           personne, Model model) {
                personneRepository.save(personne);
                return "redirect:/showPersonnes";
        }
}
```

Spring Boot

Explication

- `@ModelAttribute` permet de récupérer les valeurs ajoutées au `Model` avec `model.addAttribute("personne", new Personne());` du `GET`.
- On peut initialiser nos champs en attribuant des valeurs aux attributs de l'objet `personne` envoyé dans `model`

Spring Boot

Pour utiliser les formulaires **Spring** (comme pour la **JSTL**)

- on déclare un préfixe
- on utilise les attributs de ce préfixe pour la construction et la validation de formulaires

Spring Boot

Pour utiliser les formulaires **Spring** (comme pour la **JSTL**)

- on déclare un préfixe
- on utilise les attributs de ce préfixe pour la construction et la validation de formulaires

Déclarer le préfixe

```
<%@ taglib uri="http://www.springframework.org/tags/
   form" prefix="form" %>
```

Spring Boot

Pour utiliser les formulaires **Spring** (comme pour la **JSTL**)

- on déclare un préfixe
- on utilise les attributs de ce préfixe pour la construction et la validation de formulaires

Déclarer le préfixe

```
<%@ taglib uri="http://www.springframework.org/tags/
   form" prefix="form" %>
```

Utiliser le préfixe

```
<form:element [attributs]> ... </form:element>
```

Spring Boot

Contenu de `addPersonne.jsp`

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Ajouter une nouvelle personne</title>
</head>
<body>
        <h2>Ajouter une nouvelle personne</h2>
        <form:form method="POST" modelAttribute="personne" action="addPersonne">
                <div>
                        <form:label path="nom">Nom</form:label>
                        <form:input path="nom" />
                </div>
                <div>
                        <form:label path="prenom">Prénom</form:label>
                        <form:input path="prenom" />
                </div>
                <input type="submit" value="Ajouter">
        </form:form>
</body>
</html>
```

Spring Boot

Explication

- Les balises (`<form:x>`) sont obligatoires pour assurer le binding contrôleur/vue.
- L'attribut `modelAttribute="personne"` du formulaire correspond à l'attribut `@ModelAttribute` du contrôleur.
- L'attribut `path="..."` doit correspondre à un getter/setter de l'attribut de modèle (ici la classe `Personne`).
- À la soumission du formulaire
  - la méthode du contrôleur annotée par `@PostMapping` sera appelée,
  - les setters de la classe `Personne` seront utilisés pour enregistrer les valeurs soumises par le formulaire dans l'objet `personne`.

Spring Boot

Pour contrôler la saisie avant d'insérer une personne dans la BD

On va :

- ajouter le module de validation de `Hibernate` dans `pom.xml`
- modifier l'entité `Personne` en rajoutant des contraintes exprimées avec des annotations
- modifier la vue pour pouvoir afficher les messages d'erreur

Spring Boot

Pour ajouter la dépendance de validation

- Faire clic droit sur le projet et aller dans `Spring > Edit Starters`
- Cocher la case `Validation` dans `I/O`

Spring Boot

Pour ajouter la dépendance de validation

- Faire clic droit sur le projet et aller dans `Spring > Edit Starters`
- Cocher la case `Validation` dans `I/O`

Ou ajouter les dépendances suivantes

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Définissons nos règles de validation dans le modèle (l'entité)

```
package org.eclipse.firstspringmvc.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.Size;

@Entity
public class Personne {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long num;
        @Size(min = 2)
        @NotEmpty(message = "le champ nom est obligatoire")
        private String nom;
        @NotEmpty(message = "le champ prénom est obligatoire")
        @Size(min = 2)
        private String prenom;
        ...
```

Spring Boot

Les annotations de validation utilisées

- `@Size` : pour indiquer la taille (le nombre de caractère) min et/ou max d'un champ (on peut aussi utiliser `@Min` et `@Max`)
- `@NotEmpty` : pour préciser qu'un champ ne peut pas être vide (à ne pas confondre avec `@NotNull` qui concerne plutôt le champ d'une table et non pas d'un formulaire))

Autres annotations

- `@Email` : vérifie si la valeur est une adresse email valide
- `@Positive` et `@PositiveOrZero` : vérifie si un champ est positif (ou resp. positif ou null) (Réciproquement pour `@Negative` et `@NegativeOrZero`)
- `@Past` : vérifie si une date est dans le passé (Pareillement pour `@Future`, `@FutureOrPresent` et `@PastOrPresent`)
- `@Digits(integer=..., fraction=...)` : vérifie si les partie entière et décimale d'un nombre respectent la précision
- `@AssertTrue` : vérifie si la valeur d'un champ est `true` (réciproquement pour `@AssertFalse`)
- `@Max(value=)` : vérifie si la valeur (nombre ou chaîne de caractères représentant un nombre) est inférieure ou égale à max (réciproquement `@Min(value=)`)
- `Length(min=, max=)` : vérifie si la longueur de la chaîne de caractères est dans l'intervalle
- `@ISBN`, `@Pattern`... (consultez les autres sur `https://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/#section-validating-bean-constraints`)

Spring Boot

Ajoutons les champs d'erreurs dans `addPersonne.jsp`

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Ajouter une nouvelle personne</title>
</head>
<body>
        <h2>Ajouter une nouvelle personne</h2>
        <form:form method="POST" modelAttribute="personne" action="addPersonne">
                <div>
                        <form:label path="nom">Nom</form:label>
                        <form:input path="nom" />
                        <form:errors path="nom"  />
                </div>
                <div>
                        <form:label path="prenom">Prénom</form:label>
                        <form:input path="prenom" />
                        <form:errors path="prenom" />
                </div>
                <input type="submit" value="Ajouter">
        </form:form>
</body>
</html>
```

Pour tester la validité de ce qui a été saisi par rapport à ce qui a été défini (`Personne`)

```
@Controller
public class PersonneController {

        @Autowired
        PersonneRepository personneRepository;

        @GetMapping("addPersonne")
        public String addPersonne(Model model) {
                model.addAttribute("personne", new Personne());
                return "addPersonne";
        }

        @PostMapping("addPersonne")
        public String addPersonne(@ModelAttribute("personne") @Valid
          Personne personne, BindingResult result, Model model) {
                if(result.hasErrors()) {
                        return "addPersonne";
                }
                personneRepository.save(personne);
                return "redirect:/showPersonnes";
        }
}
```

Pour modifier le CSS de nos messages d'erreur

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Ajouter une nouvelle personne</title>
<style>
.error {
        color: red;
        font-weight: bold;
}
</style>
</head>
<body>
        <h2>Ajouter une nouvelle personne</h2>
        <form:form method="POST" modelAttribute="personne" action="addPersonne">
                <div>
                        <form:label path="nom">Nom</form:label>
                        <form:input path="nom" />
                        <form:errors path="nom" cssClass="error" />
                </div>
                <div>
                        <form:label path="prenom">Prénom</form:label>
                        <form:input path="prenom" />
                        <form:errors path="prenom" cssClass="error" />
                </div>
                <input type="submit" value="Ajouter">
        </form:form>
</body>
</html>
```

Spring Boot

Considérons la méthode suivante définie dans `FormController`

```
@GetMapping("/form")
public String showView(Model model) {

    model.addAttribute("sexe", "");
    model.addAttribute("message", "Hello World!");
    model.addAttribute("genre", new String [] {"homme", "femme"});
    model.addAttribute("personnes", personneRepository.findAll());
    return "formSpring";
}
```

Spring Boot

Dans la vue `formSpring`, pour créer des boutons radios

```
homme : <form:radiobutton path="genre" value="homme"/>
femme : <form:radiobutton path="genre" value="femme"/>
```

Spring Boot

Dans la vue `formSpring`, pour créer des boutons radios

```
homme : <form:radiobutton path="genre" value="homme"/>
femme : <form:radiobutton path="genre" value="femme"/>
```

On peut aussi récupérer les valeurs envoyées par le contrôleur

```
<form:radiobuttons items="${ genre }" path="sexe" />
```

Spring Boot

Dans la vue `formSpring`, pour créer des boutons radios

```
homme : <form:radiobutton path="genre" value="homme"/>
femme : <form:radiobutton path="genre" value="femme"/>
```

On peut aussi récupérer les valeurs envoyées par le contrôleur

```
<form:radiobuttons items="${ genre }" path="sexe" />
```

Ne pas confondre `radiobutton` et `radiobuttons`

Spring Boot

Pour créer des cases à cocher

```
homme : <form:checkbox path="genre" value="homme"/>
femme : <form:checkbox path="genre" value="femme"/>
```

Spring Boot

Pour créer des cases à cocher

```
homme : <form:checkbox path="genre" value="homme"/>
femme : <form:checkbox path="genre" value="femme"/>
```

On peut aussi récupérer les valeurs envoyées par le contrôleur

```
<form:checkboxes items="${ genre }" path="sexe" />
```

Spring Boot

Pour créer des cases à cocher

```
homme : <form:checkbox path="genre" value="homme"/>
femme : <form:checkbox path="genre" value="femme"/>
```

On peut aussi récupérer les valeurs envoyées par le contrôleur

```
<form:checkboxes items="${ genre }" path="sexe" />
```

Ne pas confondre `checkbox` et `checkboxes`

Spring Boot

Pour créer des listes déroulantes

```
<form:select path="personnes">
    <form:option value="-" label="--Choisir une personne
       --"/>
    <form:options items="${ personnes }" />
</form:select>
```

Spring Boot

Pour créer des listes déroulantes

```
<form:select path="personnes">
    <form:option value="-" label="--Choisir une personne
      --"/>
    <form:options items="${ personnes }" />
</form:select>
```

Pour créer des listes déroulantes avec sélection multiple

```
<form:select path="personnes" items="${ personnes }"
  multiple="true" />
```

Spring Boot

Pour créer des listes déroulantes

```
<form:select path="personnes">
    <form:option value="-" label="--Choisir une personne
      --"/>
    <form:options items="${ personnes }" />
</form:select>
```

Pour créer des listes déroulantes avec sélection multiple

```
<form:select path="personnes" items="${ personnes }"
  multiple="true" />
```

Pour consulter la liste des balises et d'attributs, aller sur
https://docs.spring.io/spring/docs/4.2.x
/spring-framework-reference/html/spring-form-tld.html

Spring Boot

Les sessions avec **Spring**

- on peut les utiliser via l'objet `WebRequest` (comme `HttpSession` de la plateforme **JEE**)
- mais on peut les utiliser aussi via les annotations

Les sessions

Pour récupérer `WebRequest` dans une méthode de contrôleur

```
public String ourMethod(..., WebRequest request) {
```

Les sessions

Pour récupérer `WebRequest` dans une méthode de contrôleur

```
public String ourMethod(..., WebRequest request) {
```

Pour enregistrer une donnée dans une session

```
request.setAttribute("perso", personne, WebRequest.SCOPE_SESSION);
```

Les sessions

Pour récupérer `WebRequest` dans une méthode de contrôleur

```
public String ourMethod(..., WebRequest request) {
```

Pour enregistrer une donnée dans une session

```
request.setAttribute("perso", personne, WebRequest.SCOPE_SESSION);
```

Pour récupérer la donnée d'une session (dans un contrôleur)

```
request.getAttribute("perso", WebRequest.SCOPE_SESSION);
```

Les sessions

Pour récupérer une donnée d'une session (dans une vue)

```
${ perso.nom }
```

Les sessions

Pour récupérer une donnée d'une session (dans une vue)

```
${ perso.nom }
```

Pour supprimer une variable session

```
request.removeAttribute("perso", WebRequest.SCOPE_SESSION);
```

Spring Boot

Avec les annotations

- on peut aussi annoter la classe par `@SessionAttributes` en précisant le nom d'une variable qui sera ajoutée à une session
- cette variable doit également être annotée par `ModelAttribute`

Exemple

```
@Controller
@SessionAttributes("perso")
public class ConnexionController {

  @Autowired
  private PersonneRepository personneRepository;

  @GetMapping("/connexion")
  public String personneForm(Model model) {
    model.addAttribute("perso", new Personne());
    return "connexion";
  }

  @PostMapping("/connexion")
  public String checkData(@ModelAttribute("perso") Personne personne, BindingResult result,
     Model model, WebRequest request) {
    List <Personne> personnes = personneRepository.findByNomAndPrenom(personne.getNom(),
       personne.getPrenom());
    if (personnes.size() > 0) {
      request.setAttribute("connected", true, WebRequest.SCOPE_SESSION);
      return "redirect:showPersonnes";
    }
    return "connexion";
  }

  @GetMapping("/deconnexion")
  public String leave(WebRequest request) {
    request.setAttribute("connected", false, WebRequest.SCOPE_SESSION);
    request.removeAttribute("perso", WebRequest.SCOPE_SESSION);
    return "redirect:connexion";
  }
```

Spring Boot

Remarque

- L'objet `perso` récupéré du formulaire sera ajouté comme une variable session
- Pour bien vérifier l'ajout de l'objet `perso` dans la session, on peut par exemple ajouter `${ perso.nom }` dans le body de `personneForm`.

Spring Boot

Contenu de `connexion.jsp`

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"
    %>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Page de connexion</title>
  </head>
  <body>
    <h2>Page de connexion</h2>
    <form:form modelAttribute="perso" action="connexion" method="POST">
      <form:label path="nom">Nom</form:label>
      <form:input path="nom"/>
      <form:label path="prenom">Prénom</form:label>
      <form:input path="prenom"/>
      <input type="submit" value="Connexion">
    </form:form>
  </body>
</html>
```

Spring Boot

Utilisons les sessions en modifiant le `body` de `showPersonnes.jsp`

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Liste des personnes</title>
</head>
<body>
        <p>
                Bonjour ${ personne.nom }, <a href="/addPersonne">Cliquer pour ajouter une
                  personne</a>
        </p>
        <h2>Liste des personnes</h2>
        <c:forEach items="${ personnes }" var="personne">
                <div>
                        <c:out value="${ personne.prenom } ${ personne.nom }" />
                        <a href="deletePersonne/${ personne.num }">Supprimer</a> <a
                                href="editPersonne/${ personne.num }">Modifier</a>
                </div>
        </c:forEach>
</body>
</html>
```

Spring Boot

Ne pas confondre `@SessionAttributes` et `@SessionAttribute`

`@SessionAttribute` permet de récupérer une variable session dans une méthode de contrôleur.

Exemple

```
@GetMapping("/info")
public String editPersonSession(@SessionAttribute("
    perso") Personne personne) {
   ...
  personne.setNom("Denzel")
   ...
```