Spring MVC : 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. Les formulaires
   - Récupérer les données d'un formulaire dans un objet
   - Valider les formulaires
2. Autres balises de formulaires Spring
3. Les sessions

Les formulaires

Un formulaire

- n'est pas un composant propre à Spring
- peut donc être défini comme on le faisait avant, avec HTML
- peut être construit en respectant les spécificités de Spring afin de faciliter le contrôle de la saisie et la validation

Les formulaires

Un formulaire

- n'est pas un composant propre à Spring
- peut donc être défini comme on le faisait avant, avec HTML
- peut être construit en respectant les spécificités de Spring afin de faciliter le contrôle de la saisie et la validation

L'apport de **Spring** :

- Facilite la récupération de données après la soumission d'un formulaire sous forme d'un objet
- Simplifie la validation de formulaires

Les formulaires

Pour récupérer les données d'un formulaire dans un objet

On doit :

- construire le formulaire en utilisant les balises **Spring**
- utiliser l'annotation `@ModelAttribute` pour récupérer les données envoyées par le formulaire sous forme d'objet

Les formulaires

Pour récupérer les données d'un formulaire dans un objet

On doit :

- construire le formulaire en utilisant les balises **Spring**
- utiliser l'annotation `@ModelAttribute` pour récupérer les données envoyées par le formulaire sous forme d'objet

Pour cela

On va :

- créer un contrôleur `FormController`
- créer une vue `personneForm.jsp` et construire le formulaire en utilisant les balises **Spring**

Les formulaires

Le contrôleur `FormController`

```
package org.eclipse.FirstSpringMvc.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import org.eclipse.FirstSpringMvc.dao.PersonneRepository;
import org.eclipse.FirstSpringMvc.model.Personne;

@Controller
public class FormController {

  @Autowired
  private PersonneRepository personneRepository;
```

Les formulaires

Le contrôleur `FormController` (suite)

```
  @GetMapping("/personne")
  public String personneForm(Model model) {
    model.addAttribute("personne", new Personne());
    return "personneForm";
  }
  @PostMapping("/personne")
  public String personneSubmit(@ModelAttribute("personne")
     Personne personne, Model model) {
    Personne p1 = personneRepository.save(personne);
    model.addAttribute("personne",p1);
    return "success";
  }
}
```

`@ModelAttribute` permet de matcher les valeurs de notre formulaire avec un objet de notre modèle (ici `Personne`). Ceci est possible grâce au `model.addAttribute("personne", new Personne());` du `get`.

On peut initialiser nos champs en attribuant des valeurs aux attributs de l'objet `personne` envoyé par le `model`

Les formulaires

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

- déclarer un préfixe
- les attributs de ce préfixe pour faciliter par la suite la validation de formulaires

Les formulaires

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

- déclarer un préfixe
- les attributs de ce préfixe pour faciliter par la suite la validation de formulaires

Déclarer le préfixe

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

Les formulaires

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

- déclarer un préfixe
- les attributs de ce préfixe pour faciliter par la suite 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>
```

Les formulaires

Contenu du formulaire `personneForm.jsp`

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"
    %>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Person Form</title>
  </head>
  <body>
    <form:form modelAttribute="personne" action="personne" 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="Ajouter"/>
    </form:form>
  </body>
</html>
```

Les formulaires

Explication

- Les balises Spring (`<form:x>`) sont indispensables pour 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`). Lorsque le formulaire est soumis, les setters seront utilisé pour enregistrer les valeurs du formulaire dans l'objet `personne`.
- Enfin, lorsque le formulaire est soumis, la méthode du contrôleur annotée par `@PostMapping` est appelée et le formulaire affecte automatiquement les valeurs du formulaire à l'attribut `personne` du contrôleur.

Les formulaires

Créons la vue `success.jsp`

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/
       html; charset=UTF-8">
    <title>Successfull insertion</title>
  </head>
  <body>
    Person of ${ personne.num } has been
       successfully added.
  </body>
</html>
```

Les formulaires

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

Les formulaires

Ajoutons la dépendance suivante dans `pom.xml`

```
<dependency>
  <groupId>org.hibernate.validator</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>6.0.9.Final</version>
</dependency>
```

Enregistrer pour démarrer les téléchargements

Configuration du projet

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

```
package org.eclipse.FirstSpringMvc.model;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.Size;

@Entity
@Table(name = "personnes")
public class Personne implements Serializable {
        @Id @GeneratedValue
        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;
        ...
```

Les formulaires

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))

Les formulaires

Autres annotations

- `@Email` : pour indiquer qu'une adresse email doit être saisie
- `@Positive` et `@PositiveOrZero` : pour préciser qu'un champ doit être respectivement positif (ou resp. positif ou null) (Pareillement pour `@Negative` et `@NegativeOrZero`)
- `@Past` : pour indiquer qu'une date est inférieure à la date actuelle (Pareillement pour `@Future`, `@FutureOrPresent` et `@PastOrPresent`)
- `@Digits(integer=..., fraction=...)` : pour indiquer qu'un champ doit contenir un nombre dont la partie entière et décimale respecte la précision
- `@AssertTrue` : pour indiquer que la valeur d'un champ doit être `true` (pareillement pour `@AssertFalse`)
- ...

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

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"
    %>
<html>
  <head>
    <style>
      .error{color: red;}
    </style>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Person Form</title>
  </head>
  <body>
    <form:form modelAttribute="personne" action="personne" method="post
       ">
      <form:label path="nom">nom</form:label>
      <form:input path="nom"/>
      <form:errors path="nom" cssClass ="error"/>
      <form:label path="prenom">prénom</form:label>
      <form:input path="prenom" />
      <form:errors path="prenom" cssClass ="error"/>
      <input type="submit" value="Ajouter"/>
    </form:form>
  </body>
</html>
```

Les formulaires

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

```
@PostMapping("/personne")
public String personneSubmit(@ModelAttribute("personne") @Valid
    Personne personne, BindingResult result, Model model) {

  if (result.hasErrors()) {
    return "personneForm";
  }
  Personne p1 = personneRepository.save(personne);
  model.addAttribute("personne",p1);
  return "success";
}
```

`@Valid` permet de valider les valeurs saisies par rapport à ce qui a été défini dans le modèle.

Pour le binding, il faut placer le `BindingResult result` juste après le modèle.

Autres balises de formulaires Spring

Considérons le contrôleur suivant

```
@Controller
public class FormController {

  @Autowired
  private PersonneRepository personneRepository;

  @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";
  }
}
```

Autres balises de formulaires Spring

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

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

Autres balises de formulaires Spring

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" />
```

Autres balises de formulaires Spring

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`

Autres balises de formulaires Spring

Pour créer des cases à cocher

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

Autres balises de formulaires Spring

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" />
```

Autres balises de formulaires Spring

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`

Autres balises de formulaires Spring

Pour créer des listes déroulantes

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

Autres balises de formulaires Spring

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" />
```

Autres balises de formulaires Spring

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

Les sessions

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 le `WebRequest`

```
@PostMapping("/connect")
public String ourMethod(..., WebRequest request) {
```

Les sessions

Pour récupérer le `WebRequest`

```
@PostMapping("/connect")
public String ourMethod(..., WebRequest request) {
```

Pour enregister une donnée dans une session

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

Les sessions

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

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

Les sessions

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

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

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

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

Les sessions

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

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

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

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

Pour supprimer une variable session

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

Les sessions

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 être également être annotée par `ModelAttribute`

Les sessions

Exemple

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

  @Autowired
  private PersonneRepository personneRepository;

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

Les sessions

Exemple (suite)

```
@PostMapping("/connect")
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:personne";
  }
  return "connectForm";
}

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

Les sessions

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`.

Les sessions

Contenu du `connectForm.jsp`

```
<%@ taglib uri="http://www.springframework.org/tags/form"
   prefix="form" %>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset
       =UTF-8">
    <title>Connection Form</title>
  </head>
  <body>
    <form:form modelAttribute="perso" action="connect" 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>
```

Utilisons les sessions en modifions le `body` de `personneForm.jsp`

```
<body>
  Bonjour ${ perso.nom }
  <form:form modelAttribute="personne" action="personne" method="post">
    <form:label path="nom">nom</form:label>
    <form:input path="nom"/>
    <form:errors path="nom" cssClass ="error"/>
    <form:label path="prenom">prénom</form:label>
    <form:input path="prenom" />
    <form:errors path="prenom" cssClass ="error"/>
    <input type="submit" value="Ajouter"/>
  </form:form>
  <a href="deconnect" > déconnexion </a>
</body>
```

N'oublions pas d'activer les EL dans la page

```
<%@ page isELIgnored="false" %>
```

Les sessions

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

`@SessionAttribute` permet l'utilisation d'une variable session dans une méthode de contrôleur

Exemple

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