

Spring Boot : fondamentaux

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 Introduction

2 Contrôleur

- Multi-routes
- Paramètres de requête
- Variables de chemin
- Redirection

3 DevTools

4 Vue

- Appel d'une vue depuis le contrôleur
- Communication vue/contrôleur
- Référencement d'une ressource statique

5 Changement du context-path

Spring Boot

Spring Boot

- **Spring** : inclut plusieurs projets comme Spring Core, Spring Data, Spring Web (incluant le framework **Spring MVC**).
- Spring Web : projet **Spring** respectant le modèle **MVC** (**Model-View-Controller**).
- **Spring Boot** : framework permettant d'accélérer la création et la configuration de projet **Spring** (web ou autre).

Spring Boot

Spring Boot

- **Spring** : inclut plusieurs projets comme Spring Core, Spring Data, Spring Web (incluant le framework **Spring MVC**).
- Spring Web : projet **Spring** respectant le modèle **MVC** (**Model-View-Controller**).
- **Spring Boot** : framework permettant d'accélérer la création et la configuration de projet **Spring** (web ou autre).

Remarque

Pour créer un projet web avec **Spring Boot**, il faut inclure le projet **Spring Web**.

Spring Boot

Le contrôleur

- un des composants du modèle **MVC**
- une classe **Java** annotée par `@Controller` ou `@RestController`
- Il reçoit une requête du contrôleur frontal et communique avec le modèle pour préparer et retourner une réponse

Spring Boot

Remplaçons le contenu du HomeController par le code suivant :

```
package com.example.demo.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class HomeController {

    @RequestMapping(value = "/", method = RequestMethod.GET)
    public void home() {

        System.out.println("Hello World!");

    }
}
```

Spring Boot

Explication

- La première ligne indique que notre contrôleur se trouve dans le package `com.example.demo.controller`
- Les trois imports concernent l'utilisation des annotations
- L'annotation `@Controller` permet de déclarer que la classe suivante est un contrôleur **Spring**
- La valeur de l'annotation `@RequestMapping` indique la route (`/` ici) et la méthode indique le verbe **HTTP** (`GET` ici : méthode par défaut).

Spring Boot

Attributs de @RequestMapping

- path : accepte une chaîne de caractères correspondant à la route
- value : alias de path
- name : permet d'attribuer un nom à la route
- method : verbe ou méthode **HTTP**
- params : contient le tableau de paramètre accepté par la méthode
- headers : spécifie les éléments de l'en-tête
- consumes : indique le format de données accepté par la méthode
- produces : indique le format de données retourné par la méthode

Spring Boot

Depuis Spring 4, `@RequestMapping(value = "/", method = RequestMethod.GET)` **peut être remplacé par** `@GetMapping(value = "/")`

```
package com.example.demo.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class HomeController {

    @GetMapping(value = "/")
    public void home() {
        System.out.println("Hello World!");
    }
}
```

Spring Boot

Ou aussi sans préciser le nom d'attribut dans l'annotation `@GetMapping`

```
package com.example.demo.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class HomeController {

    @GetMapping("/")
    public void home() {

        System.out.println("Hello World!");
    }
}
```

Spring Boot

Pour tester

- Démarrer le serveur **Apache Tomcat**,
- Aller à l'URL `http://localhost:8080/` et vérifier qu'un Hello World! s'affiche dans la console d'**Eclipse**.

Spring Boot

Remarque

Le contrôleur peut aussi être annoté par `@RequestMapping`

```
@Controller  
@RequestMapping("/hello")  
public class HelloController {  
    ...  
}
```

Spring Boot

La méthode d'un contrôleur peut être associée à plusieurs routes

```
package com.example.demo.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class HomeController {

    @GetMapping({ "/home", "/" })
    public void home() {
        System.out.println("Hello World!");
    }
}
```

Spring Boot

La méthode d'un contrôleur peut être associée à plusieurs routes

```
package com.example.demo.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class HomeController {

    @GetMapping({ "/home", "/" })
    public void home() {
        System.out.println("Hello World!");
    }
}
```

La méthode `home` est accessible via les deux routes suivantes

- `localhost:8080/`
- `localhost:8080/hello`

Spring Boot

Paramètres de requête : paramètres ayant la forme

/chemin?param1=value1¶m2=value2

Spring Boot

Pour récupérer les paramètres de requête, on utilise l'annotation

@RequestParam (code à ajouter dans HomeController)

```
@GetMapping("/hello")
public void sayHello(@RequestParam(value = "nom") String s) {

    System.out.println("Hello " + s);
}
```

Spring Boot

Pour récupérer les paramètres de requête, on utilise l'annotation

@RequestParam (code à ajouter dans HomeController)

```
@GetMapping("/hello")
public void sayHello(@RequestParam(value = "nom") String s) {

    System.out.println("Hello " + s);
}
```

URL pour tester

localhost:8080/hello?nom=wick

Spring Boot

Explication

L'annotation `@RequestParam(value = "nom") String s` permet de récupérer la valeur du paramètre de la requête **HTTP** est de l'affecter au paramètre `s` de la méthode `sayHello()`.

Spring Boot

Explication

L'annotation `@RequestParam(value = "nom") String s` permet de récupérer la valeur du paramètre de la requête **HTTP** est de l'affecter au paramètre `s` de la méthode `sayHello()`.

Remarque

On peut aussi ajouter `params = {"nom"}` dans `@RequestMapping` pour préciser la liste des paramètres à récupérer de la requête (facultatif)

Spring Boot

Question

Peut-on accéder à `localhost:8080/hello` sans préciser le paramètre `nom` ?

© Achref EL MOUADJI

Spring Boot

Question

Peut-on accéder à `localhost:8080/hello` sans préciser le paramètre `nom` ?

Réponse

Non, une erreur sera affichée : Error 400: Required String parameter 'nom' is not present

Spring Boot

Mais, il est possible de rendre ce paramètre facultatif

```
@GetMapping("/hello")
public void sayHello(@RequestParam(value = "nom", required =
    false) String s) {

    System.out.println("Hello " + s);
}
```

Spring Boot

Mais, il est possible de rendre ce paramètre facultatif

```
@GetMapping("/hello")
public void sayHello(@RequestParam(value = "nom", required =
    false) String s) {

    System.out.println("Hello " + s);
}
```

URL pour tester

localhost:8080/hello?nom=

Spring Boot

Il est aussi possible de préciser une valeur par défaut

```
@GetMapping("/hello")
public void sayHello(@RequestParam(value = "nom", required =
    false, defaultValue = "wick") String s) {
    System.out.println("Hello " + s);
}
```

Spring Boot

Il est aussi possible de préciser une valeur par défaut

```
@GetMapping("/hello")
public void sayHello(@RequestParam(value = "nom", required =
    false, defaultValue = "wick") String s) {
    System.out.println("Hello " + s);
}
```

URL pour tester

localhost:8080/hello?nom=

Spring Boot

Si le paramètre de la requête HTTP et l'argument de la méthode portent le même nom, l'écriture peut être simplifiée

```
@GetMapping("/hello")
public void sayHello(@RequestParam String nom) {

    System.out.println("Hello " + nom);
}
```

Spring Boot

Si le paramètre de la requête HTTP et l'argument de la méthode portent le même nom, l'écriture peut être simplifiée

```
@GetMapping("/hello")
public void sayHello(@RequestParam String nom) {

    System.out.println("Hello " + nom);
}
```

URL pour tester

localhost:8080/hello?nom=wick

Spring Boot

Variables de chemin : paramètres ayant la forme

/chemin/value1/value2

Spring Boot

Pour récupérer une variable de chemin, on utilise l'annotation `@PathVariable` (code à ajouter dans `HomeController`)

```
@GetMapping("/hello/{nom}")
public void sayHelloTo(@PathVariable(name="nom") String s) {
    System.out.println("Hello " + s);
}
```

Spring Boot

Pour récupérer une variable de chemin, on utilise l'annotation `@PathVariable` (code à ajouter dans `HomeController`)

```
@GetMapping("/hello/{nom}")
public void sayHelloTo(@PathVariable(name="nom") String s) {

    System.out.println("Hello " + s);

}
```

URL pour tester

localhost:8080/hello/wick

Spring Boot

Ou en plus simple

```
@GetMapping("/hello/{nom}")
public void sayHelloTo(@PathVariable String nom) {

    System.out.println("Hello " + nom);
}
```

Spring Boot

Ou en plus simple

```
@GetMapping("/hello/{nom}")
public void sayHelloTo(@PathVariable String nom) {

    System.out.println("Hello " + nom);
}
```

URL pour tester

localhost:8080/hello/wick

Spring Boot

Deux solutions pour la redirection

- Utiliser la classe `RedirectView`,
- Ajouter le préfixe `redirect` dans la valeur renvoyée par l'action du contrôleur.

En allant à l'URL `localhost:8080/firstspringmvc/bonjour`, on est redirigé vers `localhost:8080/firstspringmvc/home`

```
@GetMapping("/bonjour")
public RedirectView accueil () {
    RedirectView rv = new RedirectView();
    rv.setUrl("home");
    return rv;
}
```

En allant à l'URL `localhost:8080/firstspringmvc/bonjour`, on est redirigé vers `localhost:8080/firstspringmvc/home`

```
@GetMapping("/bonjour")
public RedirectView accueil () {
    RedirectView rv = new RedirectView();
    rv.setUrl("home");
    return rv;
}
```

Pour rediriger vers une route avec paramètre

```
@GetMapping("/bonjour")
public RedirectView accueil () {
    RedirectView rv = new RedirectView("hello");
    rv.addStaticAttribute("nom", "dalton");
    return rv;
}
```

Spring Boot

Solution avec le préfixe redirect

```
@GetMapping("/bonjour")
public String accueil (Model model) {
    return "redirect:hello?nom=dalton";
}
```

Spring Boot

DevTools

- outil de développement
- fonctionnant en mode développement
- permettant de redémarrer le projet après chaque changement (sauvegarde)

Spring Boot

Intégrer **DevTools** sous **Eclipse**

- Faire clic droit sur le projet
- Aller à Spring > Add DevTools

© Achref EL MOUADJI

Spring Boot

Intégrer **DevTools** sous **Eclipse**

- Faire clic droit sur le projet
- Aller à Spring > Add DevTools

Ou ajouter la dépendance Maven suivante

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

Spring Boot

Avant utilisation, il faut

- vider le cache du projet
- le mettre à jour

Spring Boot

Constats

- Dans une application **MVC**, le rôle du contrôleur n'est pas d'afficher dans la console.
- C'est plutôt de communiquer avec les différents composants afin de préparer les différents éléments pour répondre à la requête utilisateur.
- Construire la réponse est le rôle d'une vue.

Spring Boot

Les vues avec **Spring**

- Permettent d'afficher des données
- Récupèrent les données envoyées par le contrôleur
- Doivent être créées dans le répertoire `WEB-INF` de `views`
- Peuvent être créées avec un simple code **HTML, JSP, JSTL** ou en utilisant un moteur de templates comme **Thymeleaf, Mustache...**

Spring Boot

Les vues avec **Spring**

- Permettent d'afficher des données
- Communiquent avec le contrôleur pour récupérer ces données
- Doivent être créées dans le répertoire `views` dans WEB-INF
- Peuvent être créées avec un simple code JSP, JSTL ou en utilisant un moteur de templates comme Thymeleaf...

Spring Boot

Par défaut

- **Spring Boot** cherche les vues dans un répertoire `webapp` situé dans `src/main`.
- Le répertoire n'existe pas, il faut le créer.

Spring Boot

Créons une première vue home.jsp dans webapp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
   prefix="c" %>
<html>
<head>
    <title>Home</title>
</head>
<body>
<h1>
    Hello world!
</h1>
</body>
</html>
```

Spring Boot

Pour afficher une vue, la méthode `home()` doit retourner son nom

```
@GetMapping({ "/home", "/" })
public String home() {
    return "home.jsp";
}
```

Spring Boot

Remarque

- En allant à l'URL
 - `localhost:8080/home` ou
 - `localhost:8080/`
- Un message d'erreur s'affiche (circular view).
- En effet, la dépendance ajoutée par **Spring Boot** pour **Apache Tomcat** sert seulement à lancer un projet web.
- Il faut donc ajouter la dépendance qui permet à **Apache Tomcat** de lire les pages **JSP**

Spring Boot

Pour la compatibilité d'*Apache Tomcat* avec les JSP, on ajoute la dépendance suivante

```
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
```

Spring Boot

Pour la compatibilité d'*Apache Tomcat* avec les JSP, on ajoute la dépendance suivante

```
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
```

Pour utiliser la JSTL, on ajoute la dépendance suivante

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
</dependency>
```

Spring Boot

Avant de tester

- Faites un clic droit sur le projet
- Allez dans Maven > Update Project...
- Relancez le projet

Spring Boot

Remarques

- On peut préciser un autre répertoire pour les vues (à placer aussi dans webapp)
- Pour éviter de préciser chaque fois l'extension et l'emplacement de la vue, on peut l'indiquer **dans** application.properties situé **dans** src/main/resources

© Achille

Spring Boot

Remarques

- On peut préciser un autre répertoire pour les vues (à placer aussi dans webapp)
- Pour éviter de préciser chaque fois l'extension et l'emplacement de la vue, on peut l'indiquer dans application.properties situé dans src/main/resources

Nouveau contenu d'application.properties

```
spring.mvc.view.prefix=/views/  
spring.mvc.view.suffix=.jsp
```

Spring Boot

Nouveau contenu de la méthode home

```
@GetMapping({ "/home", "/" })
public String home() {
    return "home";
}
```

© Achref EL MOUELLI

Spring Boot

Nouveau contenu de la méthode home

```
@GetMapping({ "/home", "/" })
public String home() {
    return "home";
}
```

N'oublions pas de déplacer `home.jsp` dans `views` qu'il faut créer dans `webapp`

Spring Boot

Nouveau contenu de la méthode home

```
@GetMapping({ "/home", "/" })
public String home() {
    return "home";
}
```

N'oublions pas de déplacer `home.jsp` dans `views` qu'il faut créer dans `webapp`

Vérifier que la vue `home.jsp` s'affiche pour

- `localhost:8080/home`
- `localhost:8080/`

Spring Boot

Question

Le contrôleur peut-il envoyer des données à la vue ?

© Achref EL MOUADJI

Spring Boot

Question

Le contrôleur peut-il envoyer des données à la vue ?

Réponse

Oui, et pour le faire, **Spring** propose plusieurs solutions.

Spring Boot

3 Solutions proposées par **Spring** pour l'envoi de données

- Model (disponible depuis **Spring 3.x**)
- ModelMap (disponible depuis **Spring 2.x**)
- ModelAndView (disponible depuis **Spring 2.x**)

Spring Boot

Première solution avec l'interface Model

```
@GetMapping(value = "/hello")
public String sayHello(
        @RequestParam String nom,
        @RequestParam String prenom,
        Model model) {

    model.addAttribute("nom", nom);
    model.addAttribute("prenom", prenom);
    return "hello";
}
```

Spring Boot

Première solution avec l'interface Model

```
@GetMapping(value = "/hello")
public String sayHello(
        @RequestParam String nom,
        @RequestParam String prenom,
        Model model) {

    model.addAttribute("nom", nom);
    model.addAttribute("prenom", prenom);
    return "hello";
}
```

Explication

On injecte l'interface Model comme paramètre de la méthode pour envoyer les attributs à la vue.

Spring Boot

Comme en JEE, on utilise EL pour récupérer les données dans la vue

```
<%@ page language="java" contentType="text/html; charset=
    UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>hello.jsp</title>
</head>
<body>
    <h1>hello.jsp</h1>
    <p>Hello ${ prenom } ${ nom }</p>
</body>
</html>
```

Exactement comme en JEE

Spring Boot

Deuxième solution avec la classe ModelMap

```
@GetMapping("/hello")
public String sayHello(
        @RequestParam String nom,
        @RequestParam String prenom,
        ModelMap model) {

    model.addAttribute("nom", nom);
    model.addAttribute("prenom", prenom);
    return "hello";
}
```

Spring Boot

Deuxième solution avec la classe ModelMap

```
@GetMapping("/hello")
public String sayHello(
        @RequestParam String nom,
        @RequestParam String prenom,
        ModelMap model) {

    model.addAttribute("nom", nom);
    model.addAttribute("prenom", prenom);
    return "hello";
}
```

Explication

On injecte la classe ModelMap comme paramètre de la méthode pour envoyer les attributs à la vue.

Spring Boot

Troisième solution avec la classe ModelAndView

```
@GetMapping("/hello")
public ModelAndView sayHello(
    @RequestParam String nom,
    @RequestParam String prenom) {

    ModelAndView mv = new ModelAndView();
    mv.addObject("nom", nom);
    mv.addObject("prenom", prenom);
    mv.setViewName("hello");
    return mv;
}
```

Spring Boot

Troisième solution avec la classe ModelAndView

```
@GetMapping("/hello")
public ModelAndView sayHello(
    @RequestParam String nom,
    @RequestParam String prenom) {

    ModelAndView mv = new ModelAndView();
    mv.addObject("nom", nom);
    mv.addObject("prenom", prenom);
    mv.setViewName("hello");
    return mv;
}
```

Explication

On instancie ou on injecte ModelAndView et on l'utilise pour retourner une seule valeur contenant les attributs et le nom de la vue.

Spring Boot

On peut spécifier le nom de la vue à l'instanciation de ModelAndView

```
@GetMapping("/hello")
public ModelAndView sayHello(
        @RequestParam String nom,
        @RequestParam String prenom) {

    ModelAndView mv = new ModelAndView("hello");
    mv.addObject("nom", nom);
    mv.addObject("prenom", prenom);
    return mv;
}
```

Spring Boot

Model vs ModelMap vs ModelAndView

- Model (**Spring 3.x**) : interface permettant d'ajouter des attributs et les passer à la vue.
- ModelMap (**Spring 2.x**) : classe implémentant l'interface Map et permettant d'ajouter des attributs sous forme de key – value et les passer à la vue. On peut donc chercher un élément selon la valeur de la clé ou de la valeur.
- ModelAndView (**Spring 2.x**) : classe contenant à la fois un ModelMap pour les attributs et un View Object. Le contrôleur pourra ainsi retourner une seule valeur.

Spring Boot

Question

La vue peut envoyer de données à une méthode du contrôleur ?

© Achref EL MOUSSA

Spring Boot

Question

La vue peut envoyer de données à une méthode du contrôleur ?

Réponse : oui

- Soit depuis un formulaire
- Soit avec un lien hypertexte

Spring Boot

Ressource statique

- Fichier de style **CSS**
- Fichier de script **JS**
- Image
- ...

Spring Boot

Appliquer un style à un paragraphe

- Dans static de src/main/resources, créer un dossier css
- Créer un fichier style.css dans le dossier css
- Référencer style.css dans une vue

© Achref El
Bachir

Spring Boot

Appliquer un style à un paragraphe

- Dans static de src/main/resources, créer un dossier css
- Créer un fichier style.css dans le dossier css
- Référencer style.css dans une vue

Contenu de style.css

```
.first {  
    color: blue;  
}
```

Spring Boot

Pour tester, référençons le fichier `style.css` et utilisons la classe CSS `first` dans `hello.jsp`

```
<%@ page language="java" contentType="text/html; charset= UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>hello.jsp</title>
    <link rel="stylesheet" href="css/style.css">
</head>
<body>
    <h1>hello.jsp</h1>
    <p class="first">Hello ${ prenom } ${ nom }</p>
</body>
</html>
```

Spring Boot

Exercice

- Créer un contrôleur PersonneController avec deux méthodes annotées respectivement par
 `@GetMapping("/addPersonne")` et
 `@PostMapping("/addPersonne")`.
- La méthode annotée par `@GetMapping("/addPersonne")` affiche la vue `addPersonne.jsp` contenant un formulaire composé de deux inputs (un pour le nom et un pour le prénom) et un bouton pour la soumission.
- La méthode annotée par `@PostMapping("/addPersonne")` récupère les données du formulaire et retourne la vue `confirm` qui affiche les données du formulaire.

Spring Boot

Correction : PersonneController

```
package com.example.demo.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class PersonneController {

    @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);
        return "confirm";
    }
}
```

Spring Boot

Correction : addPersonne.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Ajouter une nouvelle personne</title>
    </head>
<body>
    <h2>Ajouter une nouvelle personne</h2>
    <form method="POST" action="addPersonne">
        <div>
            Nom : <input type="text" name="nom">
        </div>
        <div>
            Prénom : <input type="text" name="prenom">
        </div>
        <button>Ajouter</button>
    </form>
</body>
</html>
```

Spring Boot

Correction : confirm.jsp

```
<%@ page language="java" contentType="text/html; charset=
    UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Page de confirmation</title>
</head>
<body>
    <p>${ prenom } ${ nom } a été ajoutée avec succès.</p>
</body>
</html>
```

Spring Boot

context-path

- base des routes dans les applications web
- = nom du projet dans les applications **JEE** et **Spring MVC**
- = chaîne vide dans une application **Spring Boot**

Spring Boot

context-path

- base des routes dans les applications web
- = nom du projet dans les applications **JEE** et **Spring MVC**
- = chaîne vide dans une application **Spring Boot**

Comment modifier la valeur de context-path ?

- soit dans `application.properties`
- soit dans la classe de démarrage

Solution avec application.properties

```
server.servlet.context-path=/springboot
```

© Achref EL MOUELHI ©

Solution avec application.properties

```
server.servlet.context-path=/springboot
```

Solution avec la classe de démarrage

```
@SpringBootApplication
public class FirstSpringBootApplication {
    public static void main(String[] args) {
        System.setProperty("server.servlet.context-path", "/springboot");
        SpringApplication.run(FirstSpringBootApplication.class, args);
    }
}
```

Solution avec application.properties

```
server.servlet.context-path=/springboot
```

Solution avec la classe de démarrage

```
@SpringBootApplication
public class FirstSpringBootApplication {
    public static void main(String[] args) {
        System.setProperty("server.servlet.context-path", "/springboot");
        SpringApplication.run(FirstSpringBootApplication.class, args);
    }
}
```

Pour tester, lancer l'application et aller à localhost:8080/springboot/showPersonnes.