Spring

Cours Spring Web

1. Introduction

Dans le developpement d'application web, il est necessaire de creer des applications web dynamiques, c'est a dire qui peuvent etre modifiees sans avoir a recompiler le code source. Pour cela, il existe plusieurs technologies, comme par exemple JSP, JSF, Struts, Spring, etc. Dans ce cours, nous allons nous concentrer sur Spring.

1.1. Qu'est-ce que Spring ?

Spring est un framework Java qui permet de creer des applications web. Il est compose de plusieurs modules, qui permettent de creer des applications web, des applications mobiles, des applications desktop, des applications batch, etc.

1.2. Qu'est-ce que Spring Boot ?

Spring Boot est un module de Spring qui permet de creer des applications web plus rapidement. Il permet de creer des applications web en quelques lignes de code.

2. Spring AoC

2.1. Qu'est-ce que l'AoC ?

Spring repose sur un principe de programmation appelé "Inversion de controle" ou "Inversion of control" (IoC).
Ce principe permet de decoupler les differentes couches de l'application, et de rendre l'application plus flexible.

Son principe est simple : au lieu de creer nos objets nous-memes, on va demander a Spring de creer les objets pour nous, et de nous les fournir.

2.2. Creer un projet Spring

Pour creer un projet Spring, il faut recuperer tout les fichiers jars et les mettre dans le classpath de votre projet (ou bien utiliser un IDE qui le fait pour vous).

L'utilisation de maven est aussi possible, mais nous ne verrons pas comment l'utiliser dans ce cours.

2.3. Les beans

Un bean est avant tout un objet Java. Il est cree par Spring, et il est gere par Spring.
utilisé avec le principe d'aop, nous allons avoir des objets facilement utilisables et reutilisables.

2.3.1 Les beans Spring

Spring founit des beans qui permettent une utilisation simple et rapide de l'application.

Il faut par contre enregistrer les beans dans le fichier applicationContext.xml, qui est le fichier de configuration de Spring.

2.3.2 Le fichier applicationContext.xml

Le fichier applicationContext.xml est le fichier de configuration de Spring. Il permet de configurer Spring, et de lui dire quelles classes il doit creer.

Nous le creons dans le dossier src/main/resources.
A partir de ce fichier :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd">


</beans>
```

2.3.3 Creer un bean

Pour creer un bean, il faut ajouter une balise dans le fichier applicationContext.xml.

```
<bean id="concessionnaire" class="fr.formation.concessionnaire.Concessionnaire" />
```

2.3.3 Creer un bean

La balise prend 2 attributs :

- **Id : l'identifiant du bean**
- **Class : le nom de la classe du bean**

2.3.4 Voici un exemple de fichier applicationContext.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

        <bean id="concessionnaire" class="fr.formation.concessionnaire.Concessionnaire" />
        <bean id="personne" class="fr.formation.personne.Personne" />
        <bean id="voiture" class="fr.formation.voiture.Voiture" />
</beans>
```

2.3.5 Utiliser un bean

Pour utiliser un bean il faut le recuperer depuis l'ojet ApplicationContext.

```
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

Concessionnaire concessionnaire = (Concessionnaire) context.getBean("concessionnaire");
```

2.3.6 L'utilisation des beans

L'utilisation des beans etant simplifiee, nous pouvons donc utiliser les beans dans notre application.

Seulement cette utilisation viens avec deux autres notions : l'injection de dependances et la gestion des cycles de vie.

2.4. Les injections de dépendances

L'injection de dependances est un autre principe de Spring. Il permet de creer des objets qui vont utiliser d'autres objets, sans avoir a les creer nous-memes.

Cela nous permet :

- **De creer des objets plus facilement**
- **De rendre l'application plus flexible**
- **De rendre l'application plus facile a maintenir**
- **De rendre l'application plus facile a utiliser**

2.5. Exemple

Pour illustrer ce principe, voici un exemple de code sans Spring :

```
public class Personne {
        private Voiture voiture;

        public Personne() {
                this.voiture = new Voiture();
        }
}
```

et voici le meme exemple avec Spring :

```
public class Personne {
        private Voiture voiture; <= injection d'une voiture dans une personne;
}
```

2.6. Les injections de dépendances

Il existe 3 types d'injections de dependances :

- **Par constructeur**
- **Par propriete**

2.6.1 Injection par constructeur

L'injection par constructeur permet de creer un objet en lui passant d'autres objets en parametres.

Ils s'utilisent avec la balise .

exemple :

```
<bean id="personne" class="fr.formation.personne.Personne">
        <constructor-arg ref="voiture" />
</bean>
```

2.6.2 Injection par setter

L'injection par setter permet de changer la valeur d'une propriete d'un objet.

Ils s'utilisent avec la balise .

exemple :

```
<bean id="personne" class="fr.formation.personne.Personne">
        <property name="voiture" ref="voiture" />
</bean>
```

2.5. Les scopes

Les scopes sont des attributs qui permettent de definir la portee d'un bean. Il existe 2 scopes :

- **Singleton**
- **Prototype**

2.5.1 Singleton

Le scope singleton permet de creer un seul objet pour le bean.
Cet objet est partage entre toutes les classes qui l'utilisent.

exemple :

```
<bean id="personne" class="fr.formation.personne.Personne" scope="singleton" />
```

2.5.2 Prototype

Le scope prototype permet de creer un objet pour chaque classe qui l'utilise.

exemple :

```
<bean id="personne" class="fr.formation.personne.Personne" scope="prototype" />
```

2.6 Exemple xml

```
<bean id="personne" class="com.example.Personne" scope="singleton">
        <property name="voiture" ref="voiture" />
</bean>

<bean id="voiture" class="com.example.Voiture" scope="singleton" />
```

2.6. Les annotations

Les annotations sont des attributs qui permettent de definir des informations sur un bean.

Voici existe 3 annotations principales :

- **@Component**
- **@Autowired**
- **@Scope**

2.6.1 @Component

L'annotation @Component permet de creer un bean.

C'est l'equivalent de la balise dans le fichier applicationContext.xml.

exemple :

```
@Component
public class Personne {
        private Voiture voiture;
}
```

2.6.2 @Autowired

L'annotation @Autowired permet d'injecter une dependance dans un bean.
C'est l'equivalent de la balise dans le fichier applicationContext.xml, ou de la balise .

Si l'annotation est utilisee sur un constructeur, elle permet d'injecter les dependances dans le constructeur. Si elle est utilisee sur une propriete, elle permet d'injecter la dependance dans la propriete.

exemple :

```
@Component
public class Personne {
        @Autowired
        private Voiture voiture;
}
```

2.6.3 @qualifier

L'annotation @Qualifier permet de specifier le nom du bean a injecter.

exemple :

```
@Component
public class Personne {
        @Autowired
        @Qualifier("voiture")
        private Voiture voiture;
}
```

2.6.3 @Scope

L'annotation @Scope permet de definir le scope d'un bean.

exemple :

```
@Component
@Scope("singleton")
public class Personne {
        private Voiture voiture;
}
```

2.6.4 Les annotations supplementaires

Il existe 2 annotations qui permettent de definir des actions a effectuer avant et apres l'utilisation d'un bean.

- **@PostConstruct**
- **@PreDestroy**

3. Spring MVC

Nous allons maintenant voir comment utiliser Spring MVC pour creer une application web.
Dans cette partie nous allons voir les differents elements qui composent Spring MVC.

- **Les controllers**
- **Les vues**
- **Les models**

3.1. Les controllers

Les controllers sont des classes qui permettent de gerer les requetes HTTP.
Ils seront appeles par le DispatcherServlet.
Les controllers sont des beans Spring.
Ils sont crees automatiquement par Spring.

Exemple :

```
@Controller
public class PersonneController {
        @RequestMapping("/personne")
        public String getPersonne() {
                return "personne";
        }
}
```

3.2. Les vues

Les vues sont des pages web qui seront affichees par le navigateur.
Les vues sont des fichiers .jsp.
Les vues sont stockees dans le dossier /WEB-INF/views.

Ce dossier est protege par le serveur web, il n'est pas possible d'y acceder directement.

Exemple :

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Personne</title>
</head>
<body>
        <h1>Personne</h1>
        <p>Nom : ${personne.nom}</p>
        <p>Prenom : ${personne.prenom}</p>
</body>
</html>
```

3.3. Les models

Les models sont des objets qui seront passes aux vues.

Les models sont des beans Spring.

Les models sont crees automatiquement par Spring.

Exemple :

```
@Component
public class Personne {
        private String nom;
        private String prenom;
}
```

4. Exemple

Nous allons maintenant voir un exemple d'application web qui utilise Spring MVC.

Vous allez creer une application web qui affichera Hello World.

Pour cela nous allons creer un controller, une vue et un model.

4.1. Le controller

Il faudra toujours creer un controller pour chaque page web.

Dans notre exemple nous allons creer un controller qui affichera Hello World.

```java
@Controller
public class HelloController {
        @RequestMapping("/hello")
        public String getHello() {
                return "hello";
        }
}
```

4.2. Le model

Dans notre exemple nous n'avons pas besoin de model.

4.3. La vue

Nous allons creer une vue qui affichera Hello World.
Elle devra etre stockee dans le dossier /WEB-INF/views.
Nous allons l'appeler hello.jsp.

Nous rajouterons un titre et un paragraphe qui affichera Hello World.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Hello</title>
</head>
<body>
        <h1>Hello</h1>
        <p>Hello World</p>
</body>
</html>
```

Vous venez de creer votre premiere application web avec Spring MVC. !!!

5. L'utilisation du model

Maintenant que nous avons rapidement vu les differents elements qui composent Spring MVC, nous allons voir comment utiliser le model.

L'utilisation du model

Nous voulons maintenant afficher le nom et le prenom d'une personne.

- **Nous allons creer un model Personne.**
- **Nous allons creer un controller qui renverra deux vues differentes.**
- **Nous allons creer une vue qui affichera la personne.**

5.1. Le Controller

Nous allons creer un controller qui renverra deux vues differentes.

- **La premiere vue permettra de saisir le nom et le prenom d'une personne.**
- **La deuxieme vue permettra d'afficher le nom et le prenom de la personne.**

La saisie du nom et du prenom se fera dans un formulaire.

```java
@Controller
public class PersonneController {

	@RequestMapping("/personne") // URL de notre controller
	public String getPersonne() {
		return "personne-form"; // nom de la vue
	}

	// on laisse la seconde methode vide pour l'instant
}
```

5.2. La premiere vue

Nous allons creer une premiere vue qui permettra de saisir le nom et le prenom d'une personne.

Nous allons l'appeler personne-form.jsp.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Personne</title>
</head>
<body>
        <h1>Personne</h1>
        <form action="personne" method="post">
                <p>Nom : <input type="text" name="nom" /></p>
                <p>Prenom : <input type="text" name="prenom" /></p>
                <p><input type="submit" value="Envoyer" /></p>
        </form>
</body>
</html>
```

5.5. Le controller

Nous allons maintenant modifier notre controller pour qu'il puisse nous rediriger vers la deuxieme vue.

```
@Controller
public class PersonneController {

	@RequestMapping("/personne") // URL de notre controller
	public String getPersonne() {
		return "personne-form"; // nom de la vue
	}

	@RequestMapping("/processForm") // URL de notre controller
	public String postPersonne() {
		return "personne-result"; // nom de la vue
	}
}
```

5.6. La deuxieme vue

Nous allons maintenant creer une deuxieme vue qui affichera le nom et le prenom de la personne.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Personne</title>
</head>
<body>
        <h1>Personne</h1>
        <p>Nom : ${param.nom}</p>
        <p>Prenom : ${param.prenom}</p>
</body>
</html>
```

Le lien avec le formulaire

Afin de pouvoir recuperer les donnees du formulaire, nous allons ajouter un lien entre la page d'accueil et la page de formulaire.

La balise `<a>` permet de creer un lien.

```
<a href="personne">Saisir une personne</a>
```

Le servlet

Un servlet est une classe Java qui permet de gerer les requetes HTTP.

Dans spring les servlets sont appeles des controllers.

Il peuvent avoir en parametre :

- HttpServletRequest : permet de recuperer les donnees de la requete HTTP
- Model : permet de stocker des donnees qui seront envoyees a la vue

Traitement des donnees

Le model est une classe Java qui permet d'envoyer des donnees a la vue.

Si je souhaite avoir acces aux donnees de ma requete HTTP, je peux utiliser la classe HttpServletRequest.
Pour recuperer les donnees de la requete HTTP, j'utilise la methode getParameter().

```
@RequestMapping("/personne")
public String getPersonne(HttpServletRequest request, Model model) {
        String nom = request.getParameter("nom");
        String prenom = request.getParameter("prenom");

        System.out.println("Nom : " + nom);
        System.out.println("Prenom : " + prenom);

        return "personne-form";
}
```

Le model

Nous pouvons modifier notre model afin d'ajouter des donnees.

```
@RequestMapping("/personne")
public String getPersonne(HttpServletRequest request, Model model) {
        String nom = request.getParameter("nom");
        String prenom = request.getParameter("prenom");

        model.addAttribute("nom", nom);
        model.addAttribute("prenom", prenom);

        return "personne-form";
}
```

La vue

Nous allons maintenant modifier notre vue afin d'afficher les donnees du model.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Personne</title>
</head>
<body>
        <h1>Personne</h1>
        <p>Nom : ${nom}</p>
        <p>Prenom : ${prenom}</p>
</body>
</html>
```

tp : formulaire

Reprendre le tp sur le formulaire et le modifier afin :

- De creer une String email dans un controller
- Recuperer le nom et le prenom depuis l'HtppServletRequest
- Creer un email de ce format : prenom.nom@gmail.com
- Ajouter cet email a ma vue

Les parametres de la requete

Nous pouvons recuperer les parametres de la requete de deux manieres :

- **En utilisant la methode getParameter() de la classe HttpServletRequest**
- **En utilisant l'annotation @RequestParam**

Si nous utilisons l'annotation @RequestParam, nous n'avons pas besoin de creer de variable.

```
@RequestMapping("/personne")
public String getPersonne(@RequestParam("nom") String nom, @RequestParam("prenom") String prenom, Model model) {
        model.addAttribute("nom", nom);
        model.addAttribute("prenom", prenom);

        return "personne-form";
}
```

Data binding

Le data binding permet de recuperer les donnees d'un formulaire et de les stocker dans un objet Java.
Cela permet de simplifier le code.
Une fois que les donnees sont stockees dans l'objet, nous pouvons l'appeler dans notre controller grace a l'annotation @ModelAttribute.

Pour utiliser le data binding, nous devons ajouter la classe que nous souhaitons utiliser dans notre controller.

```java
public class User {
        private String nom;
        private String prenom;
        private String email;

        public String getNom() {
                return nom;
        }

        public void setNom(String nom) {
                this.nom = nom;
        }

        public String getPrenom() {
                return prenom;
        }

        public void setPrenom(String prenom) {
                this.prenom = prenom;
        }

        public String getEmail() {
                return email;
        }

        public void setEmail(String email) {
                this.email = email;
        }
}
```


```java
@RequestMapping("/personne")
public String getPersonne(@ModelAttribute("user") User user) {
        return "personne-form";
}
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Personne</title>
</head>
<body>
        <h1>Personne</h1>
        <p>Nom : ${user.nom}</p>
        <p>Prenom : ${user.prenom}</p>
        <p>Email : ${user.email}</p>
</body>
</html>
```

tp : data binding

Faire une page formulaire de connexion a un compte utilisateur.
Vous devez creer une classe User avec les attributs suivants :

- **Nom**
- **Prenom**
- **Email**
- **Password**

Vous devez creer un controller qui va recuperer les donnees du formulaire et les stocker dans un objet User.
Vous devez ensuite afficher les donnees de l'objet User dans une vue.

Le model

Le model est un objet Java qui permet de stocker des donnees afin de les envoyer a la vue.

Pour creer un model, nous devons creer une classe Java.

```java
public class User {
	private String nom;
	private String prenom;
	private String email;

	public String getNom() {
		return nom;
	}

	public void setNom(String nom) {
		this.nom = nom;
	}

	public String getPrenom() {
		return prenom;
	}

	public void setPrenom(String prenom) {
		this.prenom = prenom;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}
}
```

Le controller

Pour utiliser le model, nous devons l'ajouter en parametre du controller.

```
@RequestMapping("/personne")
public String getPersonne(@ModelAttribute("user") User user) {
        return "personne-form";
}
```

La vue

Pour afficher les donnees du model, nous devons utiliser la syntaxe suivante :

```
<p>Nom : ${user.nom}</p>
```

tp : model

Faite un tp une pag