Spring Boot : introduction

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

Spring Boot

Spring MVC

- un des framework **Spring**
- basé sur l'**API Servlet** de **Java JEE**
- permettant de simplifier le développement d'applications web en respectant le patron de conception **MVC 2**

Spring Boot

Spring MVC

- un des framework **Spring**
- basé sur l'**API Servlet** de **Java JEE**
- permettant de simplifier le développement d'applications web en respectant le patron de conception **MVC 2**

Problèmes

- trop de dépendance à gérer (ce qui pose souvent un problème d'incompatibilité entre les versions)
- beaucoup de configuration (JPA, sécurité, contrôleurs, vues...)

Spring Boot

Spring Boot : encore de l'abstraction

Pour éviter les problèmes de **Spring MVC**, **Spring Boot** propose :

- Les démarreurs (`starter`) : un démarreur est une dépendance, contenant un paquet de dépendance, permettant de réaliser un type de projet (Web, Rest...). Ainsi, le développeur n'a plus à gérer, lui même le problème d'incompatibilité entre les versions.
- l'auto-configuration : c'est-à-dire laisser **Spring Boot** configurer le projet à partir de dépendances ajoutées par le développeur.

Spring Boot

Exemple, pour créer un projet web, il faut ajouter la dépendance `Maven` suivante :

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

Spring Boot

Exemple, pour créer un projet web, il faut ajouter la dépendance `Maven` suivante :

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

La dépendance Maven précédente est appelée Starter forme générale

```
spring-boot-starter-*
```

Spring Boot

Exemple, pour créer un projet web, il faut ajouter la dépendance `Maven` suivante :

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

La dépendance Maven précédente est appelée Starter forme générale

`spring-boot-starter-*`

Pour consulter la liste des starters, aller sur

https://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-build-systems.html

La dépendance `spring-boot-starter-web` inclut les six dépendances suivantes :

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-json</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-tomcat</artifactId>
</dependency>
<dependency>
  <groupId>org.hibernate.validator</groupId>
  <artifactId>hibernate-validator</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
</dependency>
```

Spring Boot

La dépendance `spring-boot-starter-web` permet donc de créer un projet web contenant :

- un serveur **Apache Tomcat**
- Spring Framework et Spring MVC
- les validateurs d'Hibernate
- jackson pour les données au format JSON
- ...

Spring Boot

Création de projet Spring Boot

- Aller dans `File > New > Other`
- Chercher `Spring`, dans `Spring Boot` sélectionner `Spring Starter Project` et cliquer sur `Next >`
- Saisir
  - `first-spring-boot` dans `Name`,
  - `com.example` dans `Group`,
  - `firstspringboot` dans `Artifact`
  - `com.example.demo` dans `Package`
- Cliquer sur `Next >`
- Chercher et cocher la case correspondante à `Spring Web`, choisir la version `2.3.3` puis cliquer sur `Next >`
- Valider en cliquant sur `Finish`

Spring Boot

Pourquoi a-t-on coché la case `Spring Web` à la création du projet ?

- pour ajouter la dépendance `spring-boot-starter-web`

Contenu de la section `dependencies` de `pom.xml`

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Spring Boot

Remarques

- Le package contenant le point d'entrée de notre application (la classe contenant le `public static void main`) est `com.example.demo`
- Tous les autres packages `dao`, `model`... doivent être dans le package `demo`.

Spring Boot

Le point de démarrage de l'application

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class FirstSpringBootApplication {

  public static void main(String[] args) {
    SpringApplication.run(FirstSpringBootApplication.class, args);
  }
}
```

Spring Boot

Explication

- `SpringApplication` : la classe de démarrage d'une application **Spring** et qui va créer une instance de la classe `ApplicationContext`
- `ApplicationContext` : l'interface centrale d'une application **Spring** permettant de fournir des informations de configuration à l'application.
- `@SpringBootApplication` : englobe les 3 annotations suivantes :
  - `@Configuration` : fait partie du noyau de **Spring Framework** et indique que la classe annoté peut contenir des méthodes annotées par `@Bean`. Ainsi,**Spring Container** peut traiter la classe et générer des beans qui seront utilisés par l'application.
  - `@EnableAutoConfiguration` : permet, au démarrage de **Spring**, de générer automatiquement les configurations nécessaires en fonction des dépendances ajoutées.
  - `@ComponentScan` : permet de scanner les package contenant des composants

Spring Boot

Pour exécuter

- Faire un clic droit sur le projet et aller dans `Run As` et cliquer sur `Spring Boot App`
- Ou faire un clic droit sur **la classe** `FirstSpringBootApplication` dans `Package Explorer`, aller dans `Run As` et cliquer sur `Java Application`

Spring Boot

Pour exécuter

- Faire un clic droit sur le projet et aller dans `Run As` et cliquer sur `Spring Boot App`
- Ou faire un clic droit sur **la classe** `FirstSpringBootApplication` dans `Package Explorer`, aller dans `Run As` et cliquer sur `Java Application`

La console nous indique

```
Tomcat started on port(s): 8080 (http) with context path ''
```

Spring Boot

Pour exécuter

- Faire un clic droit sur le projet et aller dans `Run As` et cliquer sur `Spring Boot App`
- Ou faire un clic droit sur **la classe** `FirstSpringBootApplication` dans `Package Explorer`, aller dans `Run As` et cliquer sur `Java Application`

La console nous indique

```
Tomcat started on port(s): 8080 (http) with context path ''
```

Pour tester

Aller à `http://localhost:8080/`

Spring Boot

Résultat : message d'erreur

- On a créé un projet web, mais on n'a aucune page **HTML** ni **JSP**.
- **Spring Boot**, comme **Spring MVC**, implémente le patron de conception **MVC 2**, donc il nous faut au moins un contrôleur et une vue.

Spring Boot

Fichier application.properties

- Fichier utilisé par **Spring Boot** pour la configuration
- Remplaçant les beans utilisés par **Spring MVC**

Spring Boot

Pour modifier le numéro du port du serveur, ajoutons la propriété suivante dans `application.properties`

```
server.port=3000
```

Spring Boot

Pour modifier le numéro du port du serveur, ajoutons la propriété suivante dans `application.properties`

```
server.port=3000
```

Pour tester

Aller à `http://localhost:8080/`

Spring Boot

Pour modifier le numéro du port du serveur, ajoutons la propriété suivante dans `application.properties`

```
server.port=3000
```

Pour tester

Aller à `http://localhost:8080/`

Liste de propriétés disponibles pour `application.properties`

`https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html`

Spring Boot

La valeur d'une propriété définie dans `application.properties` **peut être récupérée avec l'annotation** `@Value`

```
@Value("${nom.propriété}")
```

Spring Boot

La valeur d'une propriété définie dans `application.properties` **peut être récupérée avec l'annotation** `@Value`

```
@Value("${nom.propriété}")
```

Exemple : l'attribut `serverPort` **contiendra la valeur de la propriété** `server.port` **définie dans** `application.properties`

```
@Value("${server.port}")
private String serverPort;
```

Spring Boot

Une deuxième solution consiste à injecter `Environment`

```
@Autowired
Environment environment;
```

Spring Boot

Une deuxième solution consiste à injecter `Environment`

```
@Autowired
Environment environment;
```

Et ensuite utiliser la méthode `getProperty` **pour récupérer la valeur de la propriété** `server.port` **définie dans** `application.properties`

```
String serverPort = environment.getProperty("server.port");
```