Spring Boot : Thymeleaf

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

Intégrer **Thymeleaf** dans un projet **Spring Boot** sous **Eclipse**

- Faire clic droit sur le projet
- Aller à `Spring > Edit Starters`
- Cocher les cases respectives de `Thymeleaf`

Spring Boot

Intégrer **Thymeleaf** dans un projet **Spring Boot** sous **Eclipse**

- Faire clic droit sur le projet
- Aller à `Spring > Edit Starters`
- Cocher les cases respectives de `Thymeleaf`

Ou ajouter la dépendance Maven suivante

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

Spring Boot

Gestion de vues

- Créer deux répertoires : `jsp` et `thymeleaf` dans le répertoire `views` de `webapp`
- Déplacer et placer toutes les pages `JSP` dans `jsp`
- Placer les vues `Thymeleaf` dans `thymeleaf`

Spring Boot

Gestion de vues

- Créer deux répertoires : `jsp` et `thymeleaf` dans le répertoire `views` de `webapp`
- Déplacer et placer toutes les pages `JSP` dans `jsp`
- Placer les vues `Thymeleaf` dans `thymeleaf`

Configurons `application.properties`

```
spring.view.view-names=jsp/*
spring.thymeleaf.prefix=/views/
spring.thymeleaf.suffix=.html
spring.thymeleaf.view-names=thymeleaf/*
```

JSP et Thymeleaf

Dans les contrôleurs, remplacer chaque appel d'une page JSP

```
return "nomVue";
```

JSP et Thymeleaf

Dans les contrôleurs, remplacer chaque appel d'une page JSP

```
return "nomVue";
```

Par

```
return "jsp/nomVue";
```

JSP et Thymeleaf

Dans les contrôleurs, remplacer chaque appel d'une page JSP

```
return "nomVue";
```

Par

```
return "jsp/nomVue";
```


Pour appeler une page Thymeleaf

```
return "thymeleaf/nomVue";
```

Spring Boot

Pour tester, créer un contrôleur `ThymeleafController`

```
@Controller
public class ThymeleafController {

  @GetMapping(value = "/thymeleaf")
  public String displayMessage(Model model) {

    model.addAttribute("message", "Hello World!");
    return "thymeleaf/index";
  }
}
```

Spring Boot

La vue `index.html`

```
<!DOCTYPE html>
<html xmlns:th="www.thymeleaf.org">
  <head>
    <meta charset="ISO-8859-1">
    <title>First Thymeleaf Page</title>
  </head>
  <body>
    <p th:text = "${ message }"></p>
  </body>
</html>
```

Spring Boot

La vue `index.html`

```
<!DOCTYPE html>
<html xmlns:th="www.thymeleaf.org">
  <head>
    <meta charset="ISO-8859-1">
    <title>First Thymeleaf Page</title>
  </head>
  <body>
    <p th:text = "${ message }"></p>
  </body>
</html>
```

En allant à `http://localhost:8080/thymeleaf`, `Hello World!` **est affiché**

L'internationalisation (i18n)

Préciser les sources et l'encodage de messages dans `application.properties`

```
spring.messages.encoding=UTF-8
spring.messages.basename=messages
```

L'internationalisation (i18n)

Préciser les sources et l'encodage de messages dans `application.properties`

```
spring.messages.encoding=UTF-8
spring.messages.basename=messages
```

Contenu de `messages.properties` **(dans** `src/main/resources`**)**

```
welcome.text=Bonjour tout le monde
```

L'internationalisation (i18n)

Préciser les sources et l'encodage de messages dans `application.properties`

```
spring.messages.encoding=UTF-8
spring.messages.basename=messages
```

Contenu de `messages.properties` **(dans** `src/main/resources`**)**

```
welcome.text=Bonjour tout le monde
```

Contenu de `messages_en.properties` **(dans** `src/main/resources`**)**

```
welcome.text=Hello world
```

L'internationalisation (i18n)

Préciser les sources et l'encodage de messages dans `application.properties`

```
spring.messages.encoding=UTF-8
spring.messages.basename=messages
```

Contenu de `messages.properties` **(dans** `src/main/resources`**)**

```
welcome.text=Bonjour tout le monde
```

Contenu de `messages_en.properties` **(dans** `src/main/resources`**)**

```
welcome.text=Hello world
```

Dans une vue (Thymeleaf), ajouter

```
<h1 th:text = "#{ welcome.text }"></h1>
```

Créons la classe de configuration `MvcConfig` **dans** `com.example.demo.configuration`

```
@Configuration
public class MvcConfig implements WebMvcConfigurer{
  @Bean
  public LocaleResolver localeResolver() {
    SessionLocaleResolver sessionLocaleResolver = new
       SessionLocaleResolver();
    sessionLocaleResolver.setDefaultLocale(Locale.FRANCE);
    return sessionLocaleResolver;
  }

  @Bean
  public LocaleChangeInterceptor localeChangeInterceptor() {
    LocaleChangeInterceptor localeChangeInterceptor = new
       LocaleChangeInterceptor();
    localeChangeInterceptor.setParamName("language");
    return localeChangeInterceptor;
  }

  @Override
  public void addInterceptors(InterceptorRegistry registry) {
     registry.addInterceptor(localeChangeInterceptor());
  }
}
```

L'internationalisation (i18n)

En allant sur `http://localhost:8080/thymeleaf?language=en` **, le résultat est :**

```
Hello world
```

L'internationalisation (i18n)

En allant sur `http://localhost:8080/thymeleaf?language=en` **, le résultat est :**

```
Hello world
```

En allant sur `http://localhost:8080/thymeleaf?language=fr` **, le résultat est :**

```
Bonjour tout le monde
```

L'internationalisation (i18n)

En allant sur `http://localhost:8080/thymeleaf?language=en` **, le résultat est :**

```
Hello world
```

En allant sur `http://localhost:8080/thymeleaf?language=fr` **, le résultat est :**

```
Bonjour tout le monde
```


En allant sur `http://localhost:8080/thymeleaf?language=it` **, le résultat est toujours le même :**

```
Bonjour tout le monde
```