

Spring MVC : 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

- 1 Introduction
- 2 Intégration
- 3 Configuration
- 4 Un premier Hello World avec Thymeleaf
- 5 Les expressions
 - Les expressions de variable
 - Les expressions de sélection
 - Les expressions de lien
 - Les expressions de fragment
 - Les expressions de message
- 6 JSP et Thymeleaf

Introduction

Thymeleaf

- un moteur de template écrit en Java
- pouvant être utilisé dans un environnement web utilisant l'API Servlet
- étendant les EL (Expression Language) de la librairie JSTL et simplifiant encore plus l'écriture d'un contenu dynamique pour les pages web
- générant des pages d'extensions HTML, XHTML ou XML.

Intégration

Ajoutons la dépendance suivante dans `pom.xml` (dernière version stable de Thymeleaf)

```
<dependency>
  <groupId>org.thymeleaf</groupId>
  <artifactId>thymeleaf-spring5</artifactId>
  <version>3.0.11.RELEASE</version>
</dependency>
```

Intégration

Ajoutons la dépendance suivante dans `pom.xml` (dernière version stable de Thymeleaf)

```
<dependency>
  <groupId>org.thymeleaf</groupId>
  <artifactId>thymeleaf-spring5</artifactId>
  <version>3.0.11.RELEASE</version>
</dependency>
```

Enregistrer pour démarrer les téléchargements

Configuration

Étapes

- créer un bean `SpringResourceTemplateResolver` pour spécifier l'emplacement et l'extension de vues
- créer un bean `SpringTemplateEngine` pour spécifier le moteur de template à utiliser
- utiliser la méthode `configureViewResolvers` de l'interface `WebMvcConfigurer` pour modifier et personnaliser les configurations par défaut sur les vues

Configuration

Modifions la classe de configuration `MvcConfig`

```
public class MvcConfig implements WebMvcConfigurer{
    @Bean
    public SpringResourceTemplateResolver templateResolver() {
        SpringResourceTemplateResolver templateResolver = new
            SpringResourceTemplateResolver();
        templateResolver.setPrefix("/WEB-INF/views/");
        templateResolver.setSuffix(".html");
        return templateResolver;
    }
    @Bean
    public SpringTemplateEngine templateEngine() {
        SpringTemplateEngine templateEngine = new SpringTemplateEngine();
        templateEngine.setTemplateResolver(templateResolver());
        return templateEngine;
    }
    @Override
    public void configureViewResolvers(ViewResolverRegistry registry) {
        ThymeleafViewResolver resolver = new ThymeleafViewResolver();
        resolver.setTemplateEngine(templateEngine());
        registry.viewResolver(resolver);
    }
}
```

Un premier Hello World avec Thymeleaf

Créons un contrôleur `ThymeleafController`

```
@Controller
public class ThymeleafController {

    @Autowired
    private PersonRepository personneRepository;

    @GetMapping("/thymeleaf")
    public String showView(Model model) {

        model.addAttribute("message", "Hello_World!");
        model.addAttribute("personne", personneRepository.findById(1)
            .orElse(null));
        model.addAttribute("personnes", personneRepository.findAll());
        return "view";
    }
}
```


Un premier Hello World avec Thymeleaf

Pour utiliser **Thymeleaf** dans une vue

- déclarer un espace de nom `Thymeleaf`
- utiliser cet espace de nom comme attribut de balise HTML

Un premier Hello World avec Thymeleaf

Pour utiliser **Thymeleaf** dans une vue

- déclarer un espace de nom `Thymeleaf`
- utiliser cet espace de nom comme attribut de balise HTML

Créons la vue `view.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>
```

Les expressions

Cinq expressions possibles

- $\$\{ \}$: pour les expressions de variable
- $*\{ \}$: pour les expressions de sélection
- $@\{ \}$: pour les expressions de lien (`href`)
- $\sim\{ \}$: pour les expressions de fragment
- $\#\{ \}$: pour les expressions de message (`i18n`)

Les expressions

Pour afficher le contenu d'une variable définie comme attribut dans `Model`

```
<p th:text = "${_message_}"></p>  
<!-- affiche Hello World! -->
```

Les expressions

Pour afficher le contenu d'une variable définie comme attribut dans `Model`

```
<p th:text = "${_message_}"></p>  
<!-- affiche Hello World! -->
```

Pour concaténer deux chaînes

```
<p th:text = "${_message_}+_ ' _from_Marseille' "></p>  
<!-- affiche Hello World! from Marseille -->
```

Les expressions

Pour afficher le contenu d'une variable définie comme attribut dans `Model`

```
<p th:text = "${_message_}"></p>  
<!-- affiche Hello World! -->
```

Pour concaténer deux chaînes

```
<p th:text = "${_message_}+_ ' _from_Marseille' "></p>  
<!-- affiche Hello World! from Marseille -->
```

Ou aussi

```
<p th:text = "|_${_message_}_from_Marseille_|"></p>  
<!-- affiche Hello World! from Marseille -->
```

Les expressions

Il est aussi possible de combiner les deux opérateurs (Supposant que `${ personne.nom } = wick` **et** `${ personne.prenom } = john`)

```
<p th:text = "'Bonjour_' + _|${ _personne.prenom } _${ _  
  personne.nom } _| "></p>
```

```
<!-- affiche Bonjour john wick -->
```

Les expressions

Il est aussi possible de combiner les deux opérateurs (Supposant que
`${ personne.nom } = wick` **et** `${ personne.prenom } = john`)

```
<p th:text = "'Bonjour_' + _|${_personne.prenom}_|${_personne.nom}_|"></p>
```

```
<!-- affiche Bonjour john wick -->
```

Ceci déclenche une exception

```
<p th:text = "${_message}_|'_from_Marseille'"></p>
```


Les expressions

Pour faire un affichage conditionnel

```
<div th:if="{_message.length()}_>=_5_">  
  <p th:text = "{_message_}"> </p>  
</div>  
<!-- affiche Hello World! -->
```

Les expressions

Pour faire un affichage conditionnel

```
<div th:if="${_message.length()}_>=_5_">
  <p th:text = "${_message}"> </p>
</div>
<!-- affiche Hello World! -->
```

Ou aussi en utilisant le format (if) ? (then)

```
<p th:text = "${_message.length()}_>=_5_"_?_${_message}"> </p>
<!-- affiche Hello World! -->
```

Les expressions

Pour faire un affichage conditionnel que la condition soit vraie ou non

```
<div th:if="${_message.length()}_>=_15_" th:text="long"></div>  
<div th:unless="${_message.length()}_>=_15_" th:text="court"></div>  
<!-- affiche court -->
```

Les expressions

Pour faire un affichage conditionnel que la condition soit vraie ou non

```
<div th:if="${_message.length()}_>=_15_" th:text="long"></div>  
<div th:unless="${_message.length()}_>=_15_" th:text="court"></div>  
<!-- affiche court -->
```

On peut faire aussi

```
<div th:if="${_message.length()}_>=_15" th:text="long"></div>  
<div th:unless="${_message.length()}_>=_15" th:text="court"></div>  
<!-- affiche court -->
```

Les expressions

Pour faire un affichage conditionnel que la condition soit vraie ou non

```
<div th:if="{_message.length()}_>=_15_" th:text="long"></div>
<div th:unless="{_message.length()}_>=_15_" th:text="court"></div>
<!-- affiche court -->
```

On peut faire aussi

```
<div th:if="{_message.length()}_>=_15" th:text="long"></div>
<div th:unless="{_message.length()}_>=_15" th:text="court"></div>
<!-- affiche court -->
```

Ou aussi en utilisant le format (if) ? (then) : (else) (Elvis operator)

```
<div th:text="{_message.length()}_<=_15_?_'court'_:_'long' " >
</div>
<!-- affiche court -->
```

Les expressions

Pour définir une valeur par défaut (la variable `msg` n'existe pas), on utilise le format `(value) ?: (defaultValue)`

```
<p th:text = "${_msg}_?:_${_message_}"> </p>  
<!-- affiche Hello World! -->
```

Les expressions

Autres opérateurs de comparaison

- supérieur : $>$ ou `gt`
- inférieur : $<$ ou `lt`
- supérieur ou égal : \geq ou `ge`
- inférieur ou égal : \leq ou `le`

Les expressions

Autres opérateurs de comparaison

- supérieur : `>` ou `gt`
- inférieur : `<` ou `lt`
- supérieur ou égal : `>=` ou `ge`
- inférieur ou égal : `<=` ou `le`

Opérateurs d'égalité

- égal : `==` ou `eq`
- inégal : `!=` ou `neq` ou `ne`

Les expressions

Opérateurs arithmétiques

- addition : +
- soustraction : -
- multiplication : *
- division : / ou `div`
- reste de la division : % ou `mod`

Les expressions

Opérateurs arithmétiques

- addition : `+`
- soustraction : `-`
- multiplication : `*`
- division : `/` ou `div`
- reste de la division : `%` ou `mod`

Opérateurs logiques (booléens)

- et : `and`
- ou : `or`
- négation : `!` ou `not`

Les expressions

Si on a plusieurs conditions

```
<div th:switch="${_message.length()}">
  <p th:case="1">Un</p>
  <p th:case="2">Deux</p>
  <p th:case="*">Autre</p>
</div>
<!-- affiche Autre -->
```

Les expressions

Pour faire des itérations

```
<div th:each="perso_ : _${_personnes_}">
  <span th:text="${_perso.nom_}"></span>
  <span th:text="${_perso['prenom']_}"></span>
</div>
```

Les expressions

Pour faire des itérations

```
<div th:each="perso, _status_ : _${_personnes_}">
  <span th:text="${_perso.nom_}"></span>
  <span th:text="${_perso['prenom']_}"></span>
</div>
```

Et si on voudrait récupérer et afficher l'indice (comme avec la JSTL)

```
<div th:each="perso, _status_ : _${_personnes_}">
  <span th:text="${_status.index_}"></span>
  <span th:text="${_status.count_}"></span>
  <span th:text="${_perso.nom_}"></span>
  <span th:text="${_perso['prenom']_}"></span>
</div>
```

- `index` commence de 0
- `count` commence de 1

Les expressions

On peut aussi faire (le résultat est le même)

```
<div th:each="perso_:_{_personnes_}">
  <span th:text="${_persoStat.index_}"></span>
  <span th:text="${_persoStat.count_}"></span>
  <span th:text="${_perso.nom_}"></span>
  <span th:text="${_perso['prenom']_}"></span>
</div>
```

Les expressions

On peut aussi faire (le résultat est le même)

```
<div th:each="perso_ : _${_personnes_}">
  <span th:text="${_persoStat.index_}"></span>
  <span th:text="${_persoStat.count_}"></span>
  <span th:text="${_perso.nom_}"></span>
  <span th:text="${_perso['prenom']_}"></span>
</div>
```

Autres attributs de `status`

- `odd` contient `true` s'il s'agit d'un élément d'indice impair
- `even` contient `true` s'il s'agit d'un élément d'indice pair
- `first` contient `true` s'il s'agit du premier élément
- `last` contient `true` s'il s'agit du dernier élément

Les expressions

Pour déclarer une nouvelle variable (appelée `variable`)

```
<div th:with="variable=${_message.length()_}">
  <p>
    Le message de bienvenue contient
    <span th:text="${_variable_}"> </span> lettres.
  </p>
</div>
<!-- Le message de bienvenue contient 12 lettres.
-->
```


Les expressions

On peut aussi déclarer plusieurs variables et utiliser une variable déclarée dans le même bloc

```
<div th:with="variable=${_message.length()_},  
_parity=${_variable_%2_==_0_?'pair'_:_'impair'}">  
  <p>  
    Le message de bienvenue contient  
    <span th:text="${_variable_}"> </span> lettres,  
    soit un nombre  
    <span th:text="${_parity_}"> </span> de lettres  
    .  
  </p>  
</div>  
  
<!-- Le message de bienvenue contient 12 lettres,  
      soit un nombre pair de lettres.  -->
```

Les expressions

Récapitulatif

- Pour modifier le contenu textuel d'une balise, on utilise `th:text`
- Pour déclarer une variable, on utilise `th:with`
- Pour faire un test, on utilise `th:if`, `th:unless`, `th:switch`, `th:case`
- Pour itérer, on utilise `th:each`
- ...

Les expressions

Récapitulatif

- Pour modifier le contenu textuel d'une balise, on utilise `th:text`
- Pour déclarer une variable, on utilise `th:with`
- Pour faire un test, on utilise `th:if`, `th:unless`, `th:switch`, `th:case`
- Pour itérer, on utilise `th:each`
- ...

Remarque

Il existe plusieurs autres attributs **Thymeleaf** qui permet de cibler plusieurs autres attributs **HTML**

Les expressions

Autres attributs

- Pour l'attribut `class`, on peut utiliser `th:class`
- Pour les formulaires, on peut utiliser `th:action`, `th:method...`
- Pour les `input`, on peut utiliser `th:value`, `th:name...`
- ...

Les expressions

Autres attributs

- Pour l'attribut `class`, on peut utiliser `th:class`
- Pour les formulaires, on peut utiliser `th:action`, `th:method...`
- Pour les `input`, on peut utiliser `th:value`, `th:name...`
- ...

Si on ignore l'existence d'un attribut **Thymeleaf** ciblant un attribut **HTML**

On peut utiliser

```
th:attr="nomAttribut=${ valeurAttribut }"
```

Les expressions

Pour afficher le contenu d'un objet de la classe `Personne`

```
<div>  
  <span th:text="{_personne.num_}"></span>  
  <span th:text="{_personne.nom_}"></span>  
  <span th:text="{_personne.prenom_}"></span>  
</div>
```

Les expressions

Pour afficher le contenu d'un objet de la classe `Personne`

```
<div>
  <span th:text="${_personne.num}"></span>
  <span th:text="${_personne.nom}"></span>
  <span th:text="${_personne.prenom}"></span>
</div>
```

On peut aussi simplifier l'écriture en utilisant l'expression de sélection `*{ }`

```
<div th:object="${_personne}">
  <span th:text="*{_num}"></span>
  <span th:text="*{_nom}"></span>
  <span th:text="*{_prenom}"></span>
</div>
```

Les expressions

Il est aussi possible de mixer les deux `*{ }` et `${ }`

```
<div th:object="${_personne_}">
  <span th:text="${_personne.num_}"></span>
  <span th:text="*{_#object.nom_}"></span>
  <span th:text="*{_prenom_}"></span>
</div>
```


Les expressions

Il est aussi possible de mixer les deux `*{ }` et `${ }`

```
<div th:object="${_personne_}">
  <span th:text="${_personne.num_}"></span>
  <span th:text="*{_#object.nom_}"></span>
  <span th:text="*{_prenom_}"></span>
</div>
```

Si aucune sélection d'objet n'a été effectuée, `*{ }` et `${ }` sont exactement équivalentes.

```
<div>
  <span th:text="${_personne.num_}"></span>
  <span th:text="${_personne.nom_}"></span>
  <span th:text="*{_personne.prenom_}"></span>
</div>
```

Les expressions

Pour définir un lien avec Thymeleaf

```
<a th:href="@{_thymeleaf_}">Thymeleaf</a>  
<!-- En HTML, <a href=thymeleaf>Thymeleaf</a> -->
```

Les expressions

Pour définir un lien avec Thymeleaf

```
<a th:href="@{ _thymeleaf_ }">Thymeleaf</a>  
<!-- En HTML, <a href=thymeleaf>Thymeleaf</a> -->
```

Pour ajouter des paramètres de requête (Supposant que

`${ personne.nom } = wick`)

```
<a th:href="@{ _hello (nom=${ _personne.nom_ } ) _ }">Hello</a>  
<!-- En HTML, <a href="hello?nom=wick">Hello</a> -->
```

Les expressions

Pour définir un lien avec Thymeleaf

```
<a th:href="@{$_thymeleaf_}">Thymeleaf</a>  
<!-- En HTML, <a href=thymeleaf>Thymeleaf</a> -->
```

Pour ajouter des paramètres de requête (Supposant que

`${ personne.nom } = wick`)

```
<a th:href="@{$_hello(nom=${_personne.nom_})_}">Hello</a>  
<!-- En HTML, <a href="hello?nom=wick">Hello</a> -->
```

Pour ajouter des variables de chemin (Supposant que

`${ personne.nom } = wick`)

```
<a th:href="@{$_hello/{nom} (nom=${_personne.nom_})_}">  
    Hello</a>  
<!-- En HTML, <a href="hello/wick">Hello</a> -->
```

Les expressions

Considérons la vue `menu.html`

```
<!DOCTYPE html>
<html xmlns:th="www.thymeleaf.org">
<body>
    <ul>
        <li><a href="">Item 1</a>
        <li><a href="">Item 2</a>
        <li><a href="">Item 3</a>
    </ul>
</body>
</html>
```

Les expressions

Considérons la vue `menu.html`

```
<!DOCTYPE html>
<html xmlns:th="www.thymeleaf.org">
<body>
    <ul>
        <li><a href="">Item 1</a>
        <li><a href="">Item 2</a>
        <li><a href="">Item 3</a>
    </ul>
</body>
</html>
```

Pour inclure ce menu dans toutes nos pages HTML, il faut

- le définir comme un fragment
- l'insérer dans les vues là où on le souhaite

Les expressions

Pour définir le menu comme un fragment

```
<!DOCTYPE html>
<html xmlns:th="www.thymeleaf.org">
  <body>
    <div th:fragment="fragment">
      <ul>
        <li><a href="">Item 1</a>
        <li><a href="">Item 2</a>
        <li><a href="">Item 3</a>
      </ul>
    </div>
  </body>
</html>
```

Les expressions

Pour définir le menu comme un fragment

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
  <body>
    <div th:fragment="fragment">
      <ul>
        <li><a href="#">Item 1</a>
        <li><a href="#">Item 2</a>
        <li><a href="#">Item 3</a>
      </ul>
    </div>
  </body>
</html>
```

Pour inclure ce fragment dans les autres vues

```
<div th:insert="~{_menu_::_fragment_}"></div>
```


Les expressions

Ou aussi sans utiliser les expressions de fragment

```
<div th:include="menu_::_fragment"></div>
```

Les expressions

On peut aussi charger un fragment sans l'attribut `th:fragment`

```
<!DOCTYPE html>
<html xmlns:th="www.thymeleaf.org">
  <body>
    <div id="premierMenu">
      <ul>
        <li><a href="">Item 1</a>
        <li><a href="">Item 2</a>
        <li><a href="">Item 3</a>
      </ul>
    </div>
  </body>
</html>
```

Les expressions

On peut aussi charger un fragment sans l'attribut `th:fragment`

```
<!DOCTYPE html>
<html xmlns:th="www.thymeleaf.org">
  <body>
    <div id="premierMenu">
      <ul>
        <li><a href="">Item 1</a>
        <li><a href="">Item 2</a>
        <li><a href="">Item 3</a>
      </ul>
    </div>
  </body>
</html>
```

Pour inclure ce fragment dans les autres vues

```
<div th:include="~{_menu_::_#premierMenu_}"></div>
<!-- ou aussi -->
<div th:insert="~{_menu_::_#premierMenu_}"></div>
<!-- ou aussi -->
<div th:replace="~{_menu_::_#premierMenu_}"></div>
```

Les expressions

Code source généré par `th:include`

```
<div>
  <ul>
    <li><a href="">Item 1</a>
    <li><a href="">Item 2</a>
    <li><a href="">Item 3</a>
  </ul>
</div>
```

Les expressions

Code source généré par `th:include`

```
<div>
  <ul>
    <li><a href="">Item 1</a>
    <li><a href="">Item 2</a>
    <li><a href="">Item 3</a>
  </ul>
</div>
```

Code source généré par `th:insert`

```
<div>
  <div id="premierMenu">
    <ul>
      <li><a href="">Item 1</a>
      <li><a href="">Item 2</a>
      <li><a href="">Item 3</a>
    </ul>
  </div>
</div>
```

Les expressions

Code source généré par `th:include`

```
<div>
  <ul>
    <li><a href="">Item 1</a>
    <li><a href="">Item 2</a>
    <li><a href="">Item 3</a>
  </ul>
</div>
```

Code source généré par `th:insert`

```
<div>
  <div id="premierMenu">
    <ul>
      <li><a href="">Item 1</a>
      <li><a href="">Item 2</a>
      <li><a href="">Item 3</a>
    </ul>
  </div>
</div>
```

Code source généré par `th:replace`

```
<div id="premierMenu">
  <ul>
    <li><a href="">Item 1</a>
    <li><a href="">Item 2</a>
    <li><a href="">Item 3</a>
  </ul>
</div>
```

Les expressions

Les expressions de message

permettent de récupérer la valeur d'un message selon la langue choisie

Les expressions

Les expressions de message

permettent de récupérer la valeur d'un message selon la langue choisie

Étapes

- Configurer le projet en définissant des `beans` pour préciser la langue locale, l'emplacement de fichiers de message...
- Définir les fichiers de messages (une clé - une valeur par ligne)
- Utiliser les clés dans les vues pour afficher le message selon la langue choisie

Les expressions

Étapes

- créer un bean `LocaleResolver` pour déterminer les paramètres régionaux par défaut de l'application.
- créer un bean `LocaleChangeInterceptor` pour modifier les paramètres régionaux en fonction de la valeur du paramètre (de la requête) de langue ajouté à une demande.
- créer un bean `MessageSource` pour indiquer l'emplacement des fichiers de message et leurs encodages
- utiliser la méthode `addInterceptors` de l'interface `WebMvcConfigurer` pour ajouter ces beans au registre des intercepteurs de l'application.

Les expressions

Modifions la classe de configuration `MvcConfig`

```
public class MvcConfig implements WebMvcConfigurer{
    // contenu precedent
    @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;
    }

    @Bean
    public MessageSource messageSource() {
        ReloadableResourceBundleMessageSource messageSource = new
            ReloadableResourceBundleMessageSource();
        messageSource.setBasename("classpath:messages");
        messageSource.setDefaultEncoding("UTF-8");
        return messageSource;
    }

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

Les expressions

Les fichiers de messages

- Le fichier pour les paramètres régionaux par défaut (précisé dans le bean `LocaleResolver`) doit être nommé `messages.properties`
- Le fichier pour les paramètres régionaux par défaut doit être situé dans `src/main/resources`
- On peut changer son emplacement en modifiant `setBasename("classpath:OtherSource/messages")` ; dans le bean `MessageSource`
- Les fichiers pour les autres paramètres régionaux (doivent être nommés `messages_XX.properties`

Les expressions

Contenu de `messages.properties`

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

Les expressions

Contenu de `messages.properties`

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

Contenu de `messages_en.properties`

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

Les expressions

Contenu de `messages.properties`

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

Contenu de `messages_en.properties`

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

Dans une vue, ajouter

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

Les expressions

Contenu de `messages.properties`

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

Contenu de `messages_en.properties`

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

Dans une vue, ajouter

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

Pour tester les deux langues, aller à

```
http://localhost:8080/FirstSpringMvc/thymeleaf?language=en  
ou
```

```
http://localhost:8080/FirstSpringMvc/thymeleaf?language=fr  
ou
```

```
http://localhost:8080/FirstSpringMvc/thymeleaf
```

JSP et Thymeleaf

Problématique

- Les pages **JSP** ont une extension `.jsp`
- Les pages **Thymeleaf** ont une extension `.html`
- Il faut préciser les extensions dans une classe de configuration

JSP et Thymeleaf

Problématique

- Les pages **JSP** ont une extension `.jsp`
- Les pages **Thymeleaf** ont une extension `.html`
- Il faut préciser les extensions dans une classe de configuration

Comment faire ?

- Dans `views`, on crée deux répertoires : `thymeleaf` et `jsp`
- Déplacer tous les fichiers JSP dans le répertoire `jsp` et toutes les pages HTML dans `thymeleaf`
- Reconfigurer `MvcConfig`

JSP et Thymeleaf

Modifions la méthode `configureViewResolvers` dans `MvcConfig` (contenu précédent)

```
@Override
public void configureViewResolvers(ViewResolverRegistry registry) {
    ThymeleafViewResolver resolver = new ThymeleafViewResolver();
    resolver.setTemplateEngine(templateEngine());
    registry.viewResolver(resolver);
}
```

Nouveau contenu

```
@Override
public void configureViewResolvers(ViewResolverRegistry registry) {
    ThymeleafViewResolver resolver = new ThymeleafViewResolver();
    resolver.setTemplateEngine(templateEngine());
    resolver.setViewNames(new String [] {"thymeleaf/*"});
    registry.viewResolver(resolver);
}
```

JSP et Thymeleaf

Ajoutons un `bean` pour les pages JSP

```
@Bean
InternalResourceViewResolver jspViewResolver() {
    InternalResourceViewResolver viewResolver = new
        InternalResourceViewResolver();
    viewResolver.setPrefix("/WEB-INF/views/");
    viewResolver.setSuffix(".jsp");
    viewResolver.setViewNames("jsp/*");
    return viewResolver;
}
```

JSP et Thymeleaf

Dans les contrôleurs, remplacer chaque appel d'une vue

```
return "nomVue";
```

JSP et Thymeleaf

Dans les contrôleurs, remplacer chaque appel d'une vue

```
return "nomVue";
```

Par soit

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

JSP et Thymeleaf

Dans les contrôleurs, remplacer chaque appel d'une vue

```
return "nomVue";
```

Par soit

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

Ou

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