Spring Boot

Olivier Capuozzo, Frédéric Varni

Version 1.2, 2021-01-12

Table des matières

Introduction (layout avec thymleaf)

Présentation

Lorsqu'une application web est constituée de plusieurs *pages*, il est alors de bon usage de conserver une unité de représentation.

Techniquement, cette unité de représentation se base sur :

- un **modèle**, connu sous le nom de *template* ou *layout* (structuration HTML).
- un ensemble de règles CSS partagées par l'ensemble des pages
- [des dépendances de bibliothèques js]

En phase de prototypage, il est fréquent de s'appuyer sur des frameworks CSS (par exemples Bootstrap, bulma)

Ces solutions intègrent de nombreux composants CSS et JS. Ces composants sont destinés au tiers client (navigateur, smartphone…).

Il existe 2 solutions pour distribuer ces composants aux clients tiers :

- donner aux clients un lien sur le réseau de contenu (*CDN content delivery network*), pour téléchargement, par exemple http://code.jquery.com/jquery-3.3.1.js
- servir ces composants à partir du serveur web de l'application (communication HTTP entre le serveur HTTP de l'application et les clients), par exemple http://monapplication.com/js/jquery-3.3.1.js

Dans le cadre d'une application à faible diffusion, la deuxième solution peut être envisagée, de plus elle a le mérite d'assurer la disponibilité des composants (si l'application est disponible, alors ses composants le sont aussi), et, accessoirement, permet au développeur de travailler hors ligne.

webjars - une solution java

Nous nous placons dans le cas où c'est l'application qui héberge les composants dont auront besoin ses tiers clients.

Initialement, l'application devra donc les obtenir (elle utilisera CDN pour l'occasion). Elle pourrait pour cela utiliser des gestionnaires de version spécilisés comme *npm*, *yarn*, *bower*…), mais il serait dommage de ne pas profiter du gestionnaire de version *maven* ! C'est ce que permet la solution webjars, qui encapsule les composants client dans un jar. Tout cela se passe côté serveur bien entendu.

Les WebJars sont des librairies à destination des clients-web (par exemple jQuery & Bootstrap), packagées dans un fichier JAR (Java Archive), et disponible via divers gestionnaires d'automatisation de taches, comme *maven* par exemple.

https://www.webjars.org/documentation

La définition de la dépendance devra être copiée dans le fichier **pom.xml** du projet. C'est à partir de cette référence que **maven** téléchargera dans son dépot local les fichiers en question, et rendra accessibles ces fichiers en les plaçant dans le *CLASSPATH* du projet (ce qui les rendra accessible à l'IDE).

webjars - cas bootstrap

Listing 1. pom.xml

```
  [...]

 <dependency>
   <groupId>org.webjars</groupId>
   <artifactId>bootstrap</artifactId>
   <version>4.1.3</version>
 </dependency>

 <!-- https://mvnrepository.com/artifact/org.webjars/jquery -->
 <dependency>
   <groupId>org.webjars</groupId>
   <artifactId>jquery</artifactId>
   <version>3.3.1-1</version>
 </dependency>

</dependencies>
```

Ces déclarations de dépendances, lorsqu'elle seront évaluées par *maven*, provoqueront le rapatriement, sur la machine hôte, des ressources inscrites dans le *webjars*. Ces ressources seront alors accessibles au projet, via l'IDE, au même titre que des librairies java.

Le développeur pourra donc y faire références à la conception des vues.

layout - cas thymeleaf

Avec *thymeleaf*, la conception d'un modèle de présentation (**layout** ou **template**) est basée sur la décomposition du modèle en **fragments**.

Voici une organisation typique de ressources liées à la logique de présentation:

```
├──── application.properties
├──── static
│    └──── css
│        └──── main.css ①
└──── templates
    ├──── admin ②
    │    └──── index.html ③
    ├──── error
    │    └──── 403.html
   ...
    ├──── fragments
    │    ├──── footer.html ④
    │    └──── header.html
    └── index.html
```

① pour la redéfinition de règles héritées, propre à l'application

② en référence au rôle utilisateur prioritairement concerné (ici un `admin`)

③ les vues des controleurs sont placées dans un dossier portant le nom racine d'un contrôleur (AdminController)

④ les fichiers déclarant des **fragments** (parties réutilisées par les vues) sont logés dans le dossier *fragments*

Les différentes pages partagent toutes les mêmes parties `header` (parties <head> et menu) et `footer`.

Voici un exemple d'utilisation :

Listing 2. admin/index.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Page Admin</title>
    <div th:replace="fragments/header :: header-css (title='Admin')"/> ①
</head>
<body>

<div th:replace="fragments/header :: header"/> ②

<div class="container">

     <h1>Admin index</h1>

</div>
<!-- /.container -->

<div th:replace="fragments/footer:: footer"/> ③

</body>
</html>
```

① insertion du fragment *header-css* (définit dans le fichier *header.html*)

② insertion du fragment *header* (définit dans le fichier *header.html*)

③ insertion du fragment *footer* (définit dans le fichier *footer.html*)

Voyons maintenant des exemples de composants *fragment*

Listing 3. footer.html

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity4">
<head>
</head>
<body>
<div th:fragment="footer"> ①
  <div class="container">
    <footer>
        &copy; 2021 myapplication.com
    </footer>

    <script type="text/javascript"
            src="/webjars/bootstrap/4.1.3/js/bootstrap.min.js" ②
            th:src="@{/webjars/bootstrap/4.1.3/js/bootstrap.min.js}"> ③
    </script>

    <script type="text/javascript"
            src="/webjars/jquery/3.3.1-1/jquery.min.js">
            th:src="@{/webjars/jquery/3.3.1-1/jquery.min.js}"
            </script>
  </div>
</div>
</body>
</html>
```

① Un composant est en fait le body d'une page HTML particulière, comportant une déclaration de fragment.

② Ceci est un exemple (la doc thymleaf parle de *prototype*). Sous IntelliJ, pensez à utiliser la complétion automatique (ctrl+ esp) pour l'expression de chemin.

③ C'est la valeur *src* qui sera appliquée. Les expressions thymleaf de chemin ou de lien sont encadrées par `@{ ⋯ }`.

Listing 4. header.html

```
<html xmlns:th="http://www.thymeleaf.org">
<head>
   <th:block th:fragment="header-css"> ①
      <link rel="stylesheet" type="text/css"
        href="/webjars/bootstrap/4.1.3/css/bootstrap.min.css" ②
        th:href="@{/webjars/bootstrap/4.1.3/css/bootstrap.min.css}" /> ③

      <link rel="stylesheet" th:href="@{/css/main.css}" ④
        href="../static/css/main.css" />

      <title th:text="${title ?: 'Default title'}"></title> ⑤

   </th:block>
</head>
<body>
<div th:fragment="header"> ⑥
    <nav class="navbar navbar-inverse">
        <div class="container">
            <div class="navbar-header">
                <a class="navbar-brand" th:href="@{/}">Spring Boot</a>
            </div>
            <div id="navbar" class="collapse navbar-collapse">
                <ul class="nav navbar-nav">
                    <li class="active"><a th:href="@{/}">Home</a></li>
                </ul>
            </div>
        </div>
    </nav>
</div>

</body>
</html>
```

① déclaration d'un fragment nommé *header-css*

② prototype de lien bootstrap

③ expression du lien th:href

④ le fichier CSS pour les redéfinitions propres à l'application (ne pas faire référence au dossier `static`)

⑤ définition de la valeur de `<title>` avec valeur par défaut (opérateur elvis)

⑥ déclaration d'un autre fragment dans le même fichier

Controleur

Voici un exemple de mise en oeuvre, à minima :

```
// AdminController

 @GetMapping("/admin")
 public String admin() {
    return "/admin/index";
 }
```

Nous avons ici rangé les vues dans des dossiers portant le nom du rôle prioritairement concerné, un parti pris.

Documentation

- Documentation générale : thymeleaf documenataion
- Ne pas passer à côté des classes utilitaires : classes utilitaires - avec exemples
- Différentes façons d'inclure un fragment : thymeleaf:insclusion de fragments
- Paramétrer un fragment : fragment paramétré