Cours 3
Thymeleaf

HEI 2021 / 2022

Etat des lieux

- Servlets peuvent générer du code HTML via une application Java EE.
- Cette solution a quelques inconvénient, le code HTML :
  - n'est pas facile à écrire
  - n'est pas facile à lire.

```
out.println("<!DOCTYPE html>");
out.println("<html>");
out.println("<head>");
out.println("<meta charset=\"UTF-8\">");
out.println("<title>My first servlet</title>");
out.println("</head>");
out.println("<body>");
out.println("<h1>Welcome on my first servlet!</h1>");
out.println("</body>");
out.println("</html>");
```

Templates

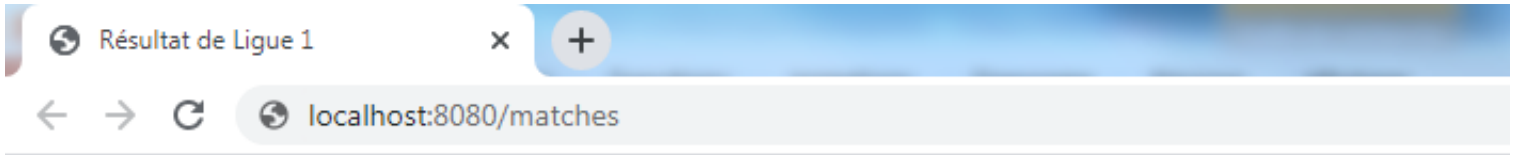
Les templates

- Une bonne méthode pour générer des pages HTML dynamiquement est d'utiliser des templates.

- Un template HTML est un fichier HTML classique qui a été enrichi par des éléments qui seront remplacés dynamiquement.

Exemple

- Imaginons que nous souhaitions créer la page suivante :



Résultat de la ligue 1 pour la journée 35 (03/05/19)

Equipe 1	Résultat		Equipe 2
Strasbourg	1	1	Marseille
PSG	1	1	Nice
Reims	0	3	Nîmes
Bordeaux	0	1	Angers
Guingamp	0	0	Caen
Nantes	3	0	Dijon
Toulouse	2	2	Rennes
Montpellier	1	1	Amiens
Monaco	2	3	Saint-Etienne
Lyon	2	2	Lille

Exemple

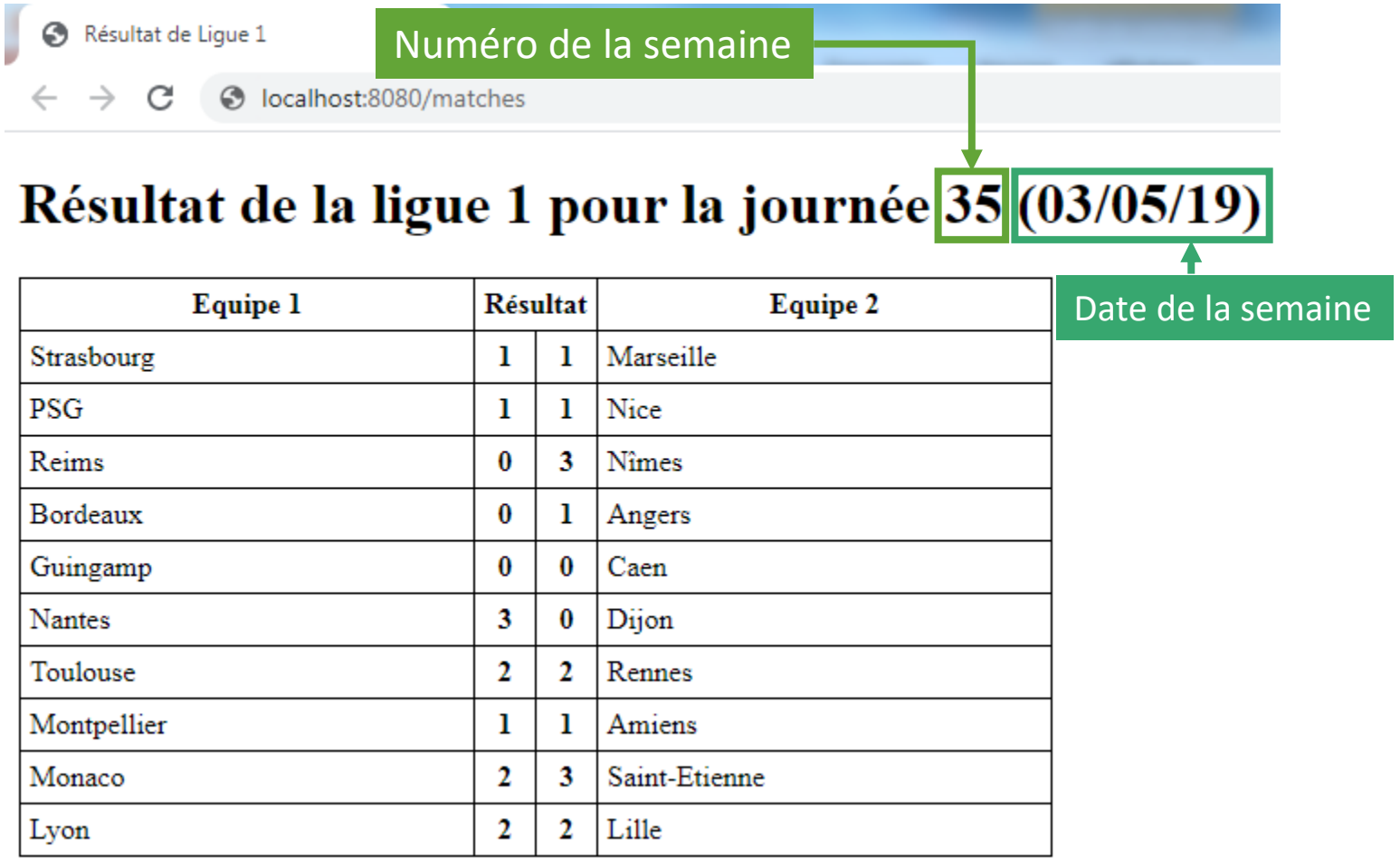
```
<!DOCTYPE html>
<html>
<head>
    <title>Résultat de Ligue 1</title>
    <meta charset="iso-8859-1"/>
    <link rel="stylesheet" href="css/matches.css" />
</head>
<body>
<h1>Résultat de la ligue 1 pour la journée 35 (03/05/19)</h1>

<table>
    <tr>
        <th scope="col" class="teamColumn">Equipe 1</th>
        <th scope="col" colspan="2">Résultat</th>
        <th scope="col" class="teamColumn">Equipe 2</th>
    </tr>
    <tr>
        <td>Strasbourg</td>
        <td class="result">1</td>
        <td class="result">1</td>
        <td>Marseille</td>
    </tr>
    <tr>
        <td>PSG</td>
```

Exemple

- Nous voulons dynamiser les données suivantes :

Matchs et résultats



Equipe 1	Résultat		Equipe 2
Strasbourg	1	1	Marseille
PSG	1	1	Nice
Reims	0	3	Nîmes
Bordeaux	0	1	Angers
Guingamp	0	0	Caen
Nantes	3	0	Dijon
Toulouse	2	2	Rennes
Montpellier	1	1	Amiens
Monaco	2	3	Saint-Etienne
Lyon	2	2	Lille

Exemple

```
<!DOCTYPE html>
<html>
<head>
    <title>Résultat de Ligue 1</title>
    <meta charset="iso-8859-1"/>
    <link rel="stylesheet" href="css/matches.css" />
</head>
<body>
<h1>Résultat de la ligue 1 pour la journée [journee] ([date])</h1>
<table>
    <tr>
        <th class="teamColumn">Equipe 1</th>
        <th colspan="2">Résultat</th>
        <th class="teamColumn">Equipe 2</th>
    </tr>
  [[FOREACH match in matches]]
    <tr>
        <td>[[match.equipe1]]</td>
        <td class="result">[[match.equipe1Score]]</td>
        <td class="result">[[match.equipe2Score]]</td>
        <td>[[match.equipe2]]</td>
    </tr>
  [[END FOREACH]]
</table>
</body>
</html>
```

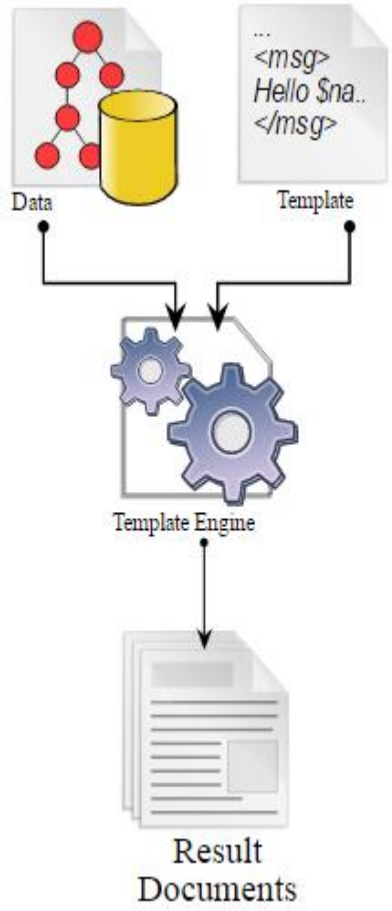
Variables

Boucle

Moteur de template

- Un moteur de template est un outil qui prend en entrée un template et des données pour générer un document final.

- Dans note cas le document final est une page HTML.



Exemple

- Dans notre exemple, les données seraient :
  - weekNumber = 35
  - weekDate = 03/05/19
  - matches = {
    Match(team1 = Strasbourg ; team1Score = 1
    team2 = Marseille ; team2Score = 1),
    Match(team1 = PSG ; team1Score = 1
    team2 = Nice; team2Score = 1),
    …
    }

Java EE et templates

- Java EE fourni quelques solution « officielles » pour créer des pages HTML dynamiques :
  - JavaServer Pages (JSP) & Standard Tag Library (JSTL) : la solution historique
  - JavaServer Faces (JSF) & Facelets : la solution la plus récente

JSP & JSTL

- JSP / JSTL est une solution assez simple pour créer des page HTML dynamique en Java EE

- Le développement de cette solution est arrêté depuis 2011 et n'évoluera probablement plus dans le futur.

JSF & Facelets

- JSF est la solution la plus logique lorsque l'on souhaite utiliser un moteur de template dans une application Java EE.

- JSF est un framework 'orienté composant' et les templates utilisé sont assez éloignés du HTML classique.

Thymeleaf 

Thymeleaf

- Thymeleaf est un moteur de template Java qui permet la génération de documents HTML.

- Il tend à remplacer les JSP dans les web applications Java et s'utilise très facilement avec les servlets.

Dépendance Maven

- Pour utiliser Thymeleaf, il faut ajouter les dépendances suivantes dans le fichier *pom.xml* :

Lib principale de Thymeleaf

```
<dependency>
   <groupId>org.thymeleaf</groupId>
   <artifactId>thymeleaf</artifactId>
   <version>3.0.9.RELEASE</version>
</dependency>
```

Lib Time

```
<dependency>
   <groupId>org.thymeleaf.extras</groupId>
   <artifactId>thymeleaf-extras-java8time</artifactId>
   <version>3.0.1.RELEASE</version>
</dependency>
```

Lib de logging

```
<dependency>
   <groupId>org.slf4j</groupId>
   <artifactId>slf4j-simple</artifactId>
   <version>1.7.25</version>
</dependency>
```

Syntaxe de base

- Thymeleaf utilise la notion de « Templates Naturels ».
  - Les templates Thymeleaf peuvent être ouvert dans un navigateur comme un simple fichier HTML et s'affichent correctement.

- Thymeleaf utilise des attributs HTML spécifiques pour gérer le templating.

Affichage de variables

- La fonctionnalité la plus basique est la dynamisation de données.
- Retour sur notre exemple :

```
<h1>Résultat de la ligue 1 pour la journée 35 (03/05/19)</h1>
```

- La dynamisation avec Thymeleaf donnerai :

```
<h1>Résultat de la ligue 1 pour la journée
   <span th:text="${weekNumber}">35</span>
   (<span th:text="${weekDate}">03/05/19</span>)
</h1>
```

Affichage de variables

- Analyse du code:

```
<h1>Résultat de la ligue 1 pour la journée
   <span th:text="${weekNumber}">35</span>
   (<span th:text="${weekDate}">03/05/19</span>)
</h1>
```

- `th:text` : remplace le contenu de la balise par la valeur démandée.
- Valeur demandée
- Valeur par défaut pour th:text

Syntaxe des expressions

Syntaxe des expressions

- Dans l'exemple précédent, le remplacement des valeurs est effectué avec l'expression suivante :

```
${weekNumber}
```

- Cette expression est appelée une **variable**.

Variables

- Les variables sont délimitées par ${…}
- Elle permettent d'accéder aux données du moteur de template Thymeleaf.
  - Ces données sont stockées dans un **contexte**.
- Le nom de la variable qui doit être récupérée est écrite entre les accolades.
  - Ce nom peut être simple ou complexe.

Variable simple

- L’exemple précédent est une variable simple :

```
${weekNumber}
```

- Cette expression va récupérer la variable `weekNumber` dans le contexte.

- L’équivalent en java est :

```
context.getVariable("weekNumber");
```

Attribut d’un objet

- Si la variable est un objet java, il est possible d’accéder à un de ses attributs via l’utilisation du point « . »

```
${person.name}
```

- Cette notation déclenchera l’appel du getter associé :

```
context.getVariable("person").getName();
```

Attribut d’un objet

- Plusieurs niveau peuvent être combinés :

```
${person.father.name}
```

- Le code java équivalent :

```
context.getVariable("person")
                    .getFather().getName();
```

Attribut d’un objet

- Les attributs d’un objet peuvent également être accédés via l’utilisation des crochets « [] » :

```
${person['father']['name']}
```

- Le code java équivalent reste inchangé :

```
context.getVariable("person")
                    .getFather().getName();
```

Variable Map

- Si la variable est une Map, les crochets permettent d'accéder aux données de la Map :

```
${personsByName['Jean Dupont'].father.name}
```

- Le code java équivalent :

```
context.getVariable("personsByName")
                    .get("Jean Dupont")
                    .getFather().getName();
```

Variable collection

- Si la variable est une Collection, les crochets permettent d'accéder aux données de celle-ci :

```
${persons[0].father.name}
```

- Le code java équivalent :

```
context.getVariable("persons").get(0)
                              .getFather().getName();
```

Appel de méthode

- Il est possible d'appeler une méthode d'objet depuis une expression :

```
${person.createCompleteName()}
${person.createCompleteNameWithSeparator('-')}
```

- Le code java équivalent :

```
context.getVariable("person").createCompleteName();
context.getVariable("person")
                .createCompleteNameWithSeparator("-");
```

Dynamisation de template

Attributs spécifiques de Thymeleaf

- Comme vu précédemment, Thymeleaf utilise des attributs spécifique pour dynamiser le HTML.

- Tous ces attributs commencent par `th:` (utilisation de la notation des espaces de nommage XML)

th:text

- Comme vu précédemment, l'attribut `th:text` remplace le contenu de la balise par sa valeur :

Thymeleaf

```
<h1>Résultat de Ligue 1 pour la journée
   <span th:text="${weekNumber}">35</span>
   (<span th:text="${weekDate}">03/05/19</span>)
</h1>
```

HTML

```
<h1>Résultat de Ligue 1 pour la journée
   <span>37</span>
   (<span>17/05/19</span>)
</h1>
```

Modification des attributs HTML

- Thymeleaf permet aussi de dynamiser les attributs des balises HTML.

- L'attribut Thymeleaf à utiliser sera la plupart du temps l'attribut HTML précédé par « `th:` »:
  - `th:value` change l'attribut `value`
  - `th:class` change l'attribut `class`
  - …

th:value

- Exemple de modification d'attribut avec `th:value`:

Thymeleaf

```
<p>
   <label for="name">Nom :</label>
   <input type="text" name="name" id="name" th:value="${person.name}">
</p>
```

HTML

```
<p>
   <label for="name">Nom :</label>
   <input type="text" name="name" id="name" value="Dupont">
</p>
```

th:alt-title

- L'attribut `th:alt-title` permet de modifier les attributs `alt` et `title` en leur positionnant la même valeur :

Thymeleaf

```
<p>
   <img th:src="${imageSrc}" th:alt-title="${imageTitle}">
</p>
```

HTML

```
<p>
   <img src="avatar/jean.png"  alt="Avatar de Jean Dupont"
        title="Avatar de Jean Dupont">
</p>
```

th:classappend

- L'attribut `th:classappend` ajoute une classe à aux classes existentes de l'élément concerné :

Thymeleaf

```
<td class="result" th:classappend="${resultClass}"  th:text="${match.scoreTeam1}"></td>
```

HTML

```
<td class="result  winner" >3</td>
```

Attributs booléens

- En HTML, certains attributs comme `checked` ou `selected` sont des booléens. Ces attributs n'ont pas de valeur.

```
<!-- is checked -->
<input type="checkbox" name="box" checked>
<!-- is not checked -->
<input type="checkbox" name="box2">
```

- L'attribut Thymeleaf équivalent **doit** avoir une valeur booléenne associée. Si la valeur est vrai, l'attribut est ajouté à l'élément.

th:checked

- th:checked est l'attribut Thymeleaf pour l'attribut booléen HTML checked:

Thymeleaf

```
<!--/* user.active is true */-->
<input type="checkbox" name="active" th:checked="${user.active}">
<!--/* user.admin is false */-->
<input type="checkbox" name="admin" th:checked="${user.admin}">
```

HTML

```
<input type="checkbox" name="active" checked >
<input type="checkbox" name="admin" >
```

Itérations

Itérations

- Un besoin classique dans un template est d'avoir à itérer sur une liste de valeurs.
- L'attribut Thymeleaf pour cette fonctionnalité est `th:each` :

```
th:each="match : ${matches}"
```

- Le code java équivalent :

```
for (Match match : context.getVariable("matches")) {

}
```

Exemple de th:each

Thymeleaf

```
<tr th:each="match : ${matches}">
    <td th:text="${match.team1}"></td>
    <td class="result" th:text="${match.scoreTeam1}"></td>
    <td class="result" th:text="${match.scoreTeam2}"></td>
    <td th:text="${match.team2}"></td>
</tr>
```

HTML

```
<tr>
    <td>Strasbourg</td>
    <td class="result">1</td>
    <td class="result">1</td>
    <td>Marseille</td>
</tr>
<tr>
    <td>PSG</td>
    <td class="result">1</td>
    <td class="result">1</td>
    <td>Nice</td>
</tr>
…
```

Valeurs Itérables

- Les objets suivants peuvent s'utiliser avec `th:each`:
  - objet implémentant l'interface `java.util.Iterable`
  - objet implémentant l'interface `java.util.Iterator`
  - objet implémentant l'interface `java.util.Map` (itération sur les `Entry` de la map)
  - les tableaux

- Tous autres objets passé à un `th:each` sera considéré comme une liste d'un seul élément contenant lui-même.

Statut d'itération

- Lors d'une itération `th:each`, une variable est créée en suffixant le nom de la variable itérée par `Stat`.

- Cette variable contient des informations sur l'itération comme :
  - Indexes : `index` (commence à 0) et `count` (#elts traités, commence à 1)
  - Nombre d'éléments : `size`
  - Information sur la position : `even`, `odd`, `first`, `last`

Exemple d’utilisation

Thymeleaf

```
<tr th:each="match : ${matches}"
    th:class="${matchStat.even} ? 'even' : 'odd'">
    <td th:text="${matchStat.count}+'/'+${matchStat.size}"></td>
    <td th:text="${match.team1}"></td>
    <td class="result"  th:text="${match.scoreTeam1}"></td>
    <td class="result"  th:text="${match.scoreTeam2}"></td>
    <td th:text="${match.team2}"></td>
</tr>
```

HTML

```
<tr class="odd">
    <td>1/10</td>
    <td>Strasbourg</td>
    <td class="result">1</td>
    <td class="result">1</td>
    <td>Marseille</td>
</tr>
<tr class="even">
    <td>2/10</td>
    <td>PSG</td>
    <td class="result">1</td>
    <td class="result">1</td>
    <td>Nice</td>
</tr>
…
```

Conditions

Conditions

- Un autre besoin classique lors de l'utilisation de template est de pouvoir faire des conditions.

- L'attribut Thymeleaf pour cette fonctionnalité est `th:if :`

```
th:if="${match.scoreTeam1 > match.scoreTeam2}"
```

- L'élément HTML contenant cet attribut (et ses fils) ne sera affiché que si la condition est évaluée à true.

Exemple de th:if

Thymeleaf

```
<td>
    <span th:if="${match.scoreTeam1 > match.scoreTeam2}">
                L'équipe 1 a gagné</span>
    <span th:if="${match.scoreTeam1 < match.scoreTeam2}">
                L'équipe 2 a gagné</span>
    <span th:if="${match.scoreTeam1 == match.scoreTeam2}">Egalité</span>
</td>
```

HTML

```
<td>
    < L'équipe 1 a gagné</span>
</td>
```

Conditions

- La plupart du temps, la valeur passée à un `th:if` sera un booléen.
- Mais n'importe quel type de valeur peut être utilisée. La condition sera évaluée à true si la valeur passée est :
  - un booléen est sa valeur est true.
  - un nombre différent de 0
  - un character différent de 0
  - une String différent de "false", "off" ou "no"
  - pas un booléen, pas un nombre, pas un character, pas une String

th:unless

- Thymeleaf fourni l'opération inverse de `th:if` : `th:unless`.

- Les 2 lignes suivantes sont équivalentes :

```
<span th:unless="${condition}">Affichage conditionnel</span>

<span th:if="${not condition}"> Affichage conditionnel </span>
```

th:switch

- L'attribut Thymeleaf `th:switch` est l'équivalent du switch en java :

```
<div th:switch="${user.role}">
    <p th:case="'admin'">L'Utilisateur est administrateur</p>
    <p th:case="'manager'">L'Utilisateur est manager</p>
    <p th:case="*">L'Utilisateur est autre chose</p>
</div>
```

- L'instruction avec la valeur * est l'instruction par défaut.

Fragments

Problème

- La plupart du temps, des blocs d'HTML sont présents sur plusieurs (ou toutes) les pages d'un site.
  - Exemple: l'en-tête, le menu, le pied de page d'une page.

- Ces blocs devraient être externalisés pour éviter la répétition de code dans chaque pages.

Fragments

- Thymeleaf utilise des **fragments** pour implémenter cette fonctionnalité.

- Dans chaque template, il est possible de déclarer un fragment avec l'attribut `th:fragment`.

```
<header th:fragment="header">
    <h1>Exemple de site avec Thymeleaf</h1>
</header>
```

Utilisation de fragment

- Les attributs `th:insert` et `th:replace` servent à ajouter le contenu d'un fragment dans un autre template.

```
<header th:replace="~{layout :: header}"></header>
```

Syntaxe

- Les expressions `th:insert` et `th:replace` utilisent un type d'expression spécifique (appelé **Fragment expression**) pour décrire le fragment à utiliser.

```
"~{templatename::selector}"
```

- `selector` est le nom du fragment à utiliser
- `templatename` est le nom du template à utiliser (le fichier dans lequel est déclaré le fragment)

Exemple d'utilisation

Thymeleaf (layout.html)

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Fragments layout</title>
    <meta charset="utf-8"/>
</head>
<body>
    <header th:fragment="header">
        <h1>Exemple de site avec Thymeleaf</h1>
    </header>
</body>
</html>
```

Thymeleaf (home.html)

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Thymeleaf 3 + Servlet 3 example</title>
    <meta charset="utf-8"/>
</head>
<body>
    <header th:replace="~{layout :: header}"></header>

    <p>Bienvenu sur ce site.</p>
</body>
</html>
```

HTML

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Thymeleaf 3 + Servlet 3 example</title>
    <meta charset="utf-8"/>
</head>
<body>
    <header>
        <h1>Exemple de site avec Thymeleaf</h1>
    </header>

    <p>Bienvenu sur ce site.</p>
</body>
</html>
```

th:insert

- `th:insert` insert le fragment en tant qu'enfant de l'élément sur lequel il est appelé :

Thymeleaf

```
<p class="copyright" th:fragment="footer">
    &copy; Copyright 2020
</p>
```

Thymeleaf

```
<footer th:insert="~{layout :: footer}"></footer>
```

HTML

```
<footer>
  <p class="copyright">
    &copy; Copyright 2020
  </p>
</footer>
```

th:replace

- `th:replace` remplace l'élément sur lequel il est appelé par le contenu du fragment :

Thymeleaf

```
<p class="copyright" th:fragment="footer">
    &copy; Copyright 2020
</p>
```

Thymeleaf

```
<footer th:replace="~{layout :: footer}"></footer>
```

HTML

```
<p class="copyright">
    &copy; Copyright 2020
</p>
```

Fragment paramétré

- Il est possible d'ajouter des paramètres aux fragments :

Thymeleaf

```
<head th:fragment="htmlhead(title)">
    <title th:text="${title}">
              Fragments layout</title>
    <meta charset="iso-8859-1"/>
</head>
```

Thymeleaf

```
<head th:replace="~{layout :: htmlhead('Accueil')}">
</head>
```

HTML

```
<head>
    <title>Accueil</title>
    <meta charset="utf-8859-1"/>
</head>
```

Autres fonctionnalités

th:block

- `th:block` est la seule balise utilisée par Thymeleaf.
- Il s'agit d'un conteneur d'attributs Thymeleaf qui disparaitra :

Thymeleaf

```
<dl>
    <th:block th:each="match : ${matches}">
        <dt th:text="${matchStat.count} + '/' + ${matchStat.size}"></dt>
        <dd>
            <span th:text="${match.team1}"></span> vs.
            <span th:text="${match.team2}"></span>
        </dd>
    </th:block>
</dl>
```

HTML

```
<dl>
    <dt>1/10</dt>
    <dd>
        <span>Strasbourg</span> vs. <span>Marseille</span>
    </dd>
    <dt>2/10</dt>
    <dd>
        <span>PSG</span> vs. <span>Nice</span>
    </dd>
```

Expression inline

- Il est possible d'ajouter des expressions dans un template sans les mettre en attribut.

```
<h1>Résultat de la ligue 1 pour la semaine [[${weekNumber}]]</h1>
```

- La syntaxe `[[…]]` remplace l'attribut `th:text`.

Fonctions utilitaires

- Des variables spéciales permettent d'appeler des fonctions utilitaires.

Variable	Description
`#dates`	méthodes pour les objets java.util.Date (formatage par exemple)
`#numbers`	méthodes pour formater des nombres
`#strings`	méthodes pour les objets de type String
`#lists`	méthodes pour les listes
…	

Gestion des objets LocalDate

- Java 8 a introduit l'API Time avec notamment les classes LocalDate et LocalDateTime.
- Pour pouvoir utiliser des fonctions permettant de manipuler ces objets en Thymeleaf, un Dialect doit être ajouté au moteur de template :

```
TemplateEngine templateEngine = new TemplateEngine();
templateEngine.addDialect(new Java8TimeDialect());
```

- Ce dialect ajoute l'objet `#temporals` qui fonctionne comme l'objet `#dates`.

Exemples d'utilisation

- Quelques exemples d'utilisations de fonctions utilitaires :

```
${#dates.format(date, 'dd/MMM/yyyy HH:mm')}
${#dates.year(date)}
${#temporals.format(date, 'dd/MMM/yyyy HH:mm')}
${#numbers.formatDecimal(num,3,2,'COMMA')}
${#strings.isEmpty(name)}
${#strings.contains(name,'ez')}
${#strings.startsWith(name,'Don')}
${#strings.toUpperCase(name)}
${#lists.size(list)}
${#lists.contains(list, element)}
```

Lien avec les servlets

TemplateResolver

- La classe `TemplateResolver` permet de définir où sont stockés les templates Thymeleaf du projet.

- La classe `ServletContextTemplateResolver` est plus spécifique aux Servlets.

```
ServletContextTemplateResolver resolver =
    new ServletContextTemplateResolver(request.getServletContext());
resolver.setPrefix("/WEB-INF/templates/");
resolver.setSuffix(".html");
resolver.setTemplateMode(TemplateMode.HTML);
```

WebContext

- La classe `WebContext` est la classe qui contient toutes les variables.

```
WebContext context = new WebContext(req, resp, req.getServletContext());
context.setVariable("weekNumber", 35);
context.setVariable("weekDate", LocalDate.of(2019, 5, 3));
context.setVariable("matches", listMatches());
```

- Toutes les variables ajoutées dans le `webContext` seront accessibles dans les expressions du template.

Request attributes

- Lorsque l'on utilise l'API Servlet, pour transmettre des variables, on les ajoutes dans les attributs de l'objet **request** :

```
request.setAttribute("varName", varValue);
```

- Thymeleaf récupère ces attributs et les ajoute à son contexte :

${varName}

Attributs de session

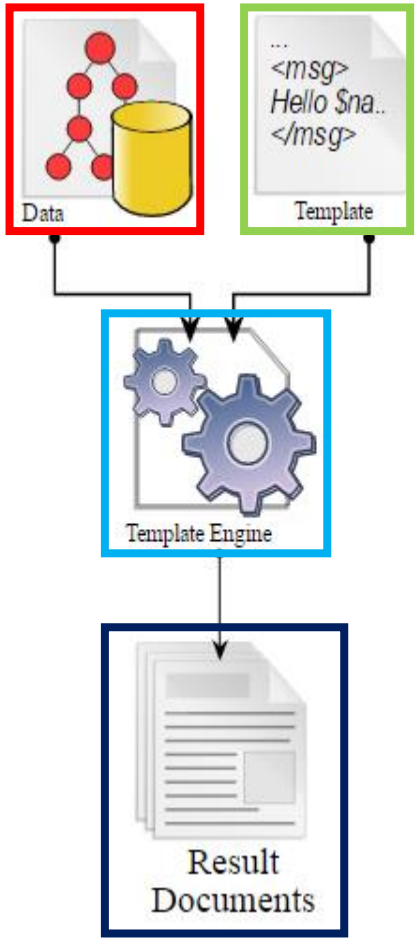
- La variable spéciale `#session` permet d'accéder aux attributs de la session :

```
${#session.user.name}
```

Le moteur de template

- La classe TemplateEngine effectue le travail de templating, elle combine le template avec les variables afin de générer le document final :

```
TemplateEngine engine = new TemplateEngine();
engine.setTemplateResolver(resolver);
String finalDocument =
        engine.process("index", webContext);
resp.getWriter().write(finalDocument);
```



Le moteur de template

- `process()` est la méthode qui effectue tout le travail :

```
String finalDocument = engine.process("index", webContext);
```

- index: nom du fichier template
- webContext: variables qui serviront à dynamiser le template
- finalDocument: fichier HTML final

Le moteur de template

- Il est possible pour le moteur de template d'écrire directement dans la réponse HTTP :

```
engine.process("index", webContext, resp.getWriter());
```

Conclusion

- La méthode `doGet()` ressemblera finalement à ça :

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                        throws ServletException, IOException {
    ServletContextTemplateResolver resolver =
                    new ServletContextTemplateResolver(req.getServletContext());
    resolver.setPrefix("/WEB-INF/templates/");
    resolver.setSuffix(".html");
    resolver.setTemplateMode(TemplateMode.HTML);

    TemplateEngine engine = new TemplateEngine();
    engine.setTemplateResolver(resolver);

    WebContext context = new WebContext(req, resp, req.getServletContext());
    context.setVariable("weekNumber", 35);
    context.setVariable("weekDate", LocalDate.of(2019, 5,3));
    context.setVariable("matches", listMatches());
    engine.process("matches", context, resp.getWriter());
}
```