Spring Boot 2.x Thymeleaf

By

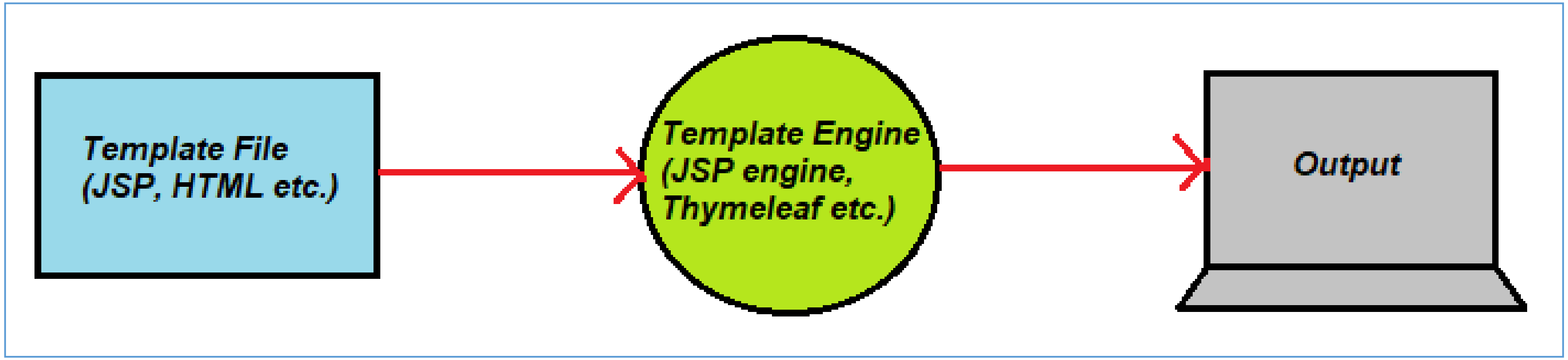
Anand Kulkarni

Table of Content

Module	**Topic**
Module 1:	Introduction to Thymeleaf
Module 2:	Thymeleaf vs JSP
Module 3:	Thymeleaf setup
Module 4:	Thymeleaf template
Module 5:	Standard expression syntax
Module 6:	Loop iteration & conditional statements

What is a Template Engine?

Template engine is a program that processes the template file & produces the required output.



What is a Thymeleaf?

- Thymeleaf is a modern server-side Java template engine for both web and standalone environments.
- Thymeleaf's main goal is to bring elegant natural templates to your development workflow.

JSP & Thymeleaf – Similarity & Differences

JSP	Thymeleaf
JSP is a view layer in Spring MVC.	Thymeleaf is a view layer in Spring MVC.
JSP engine converts the JSP page into a servlet which is associated with a specific life cycle.	Thymeleaf engine converts HTML template code into HTML output.
JSP is a old specification & there are no improvements in JSP since last many years.	Thymeleaf is a new specification & we find several improvements in Thymeleaf every month.
Java code inside JSP looks difficult to distinguish between Java & HTML code.	Thymeleaf template code provides tags those look like similar to HTML. Hence, Thymeleaf code is more readable than JSP

Thymeleaf Setup

Assuming you have already created a Spring Boot application. Please follow below steps to provide Thymeleaf support:

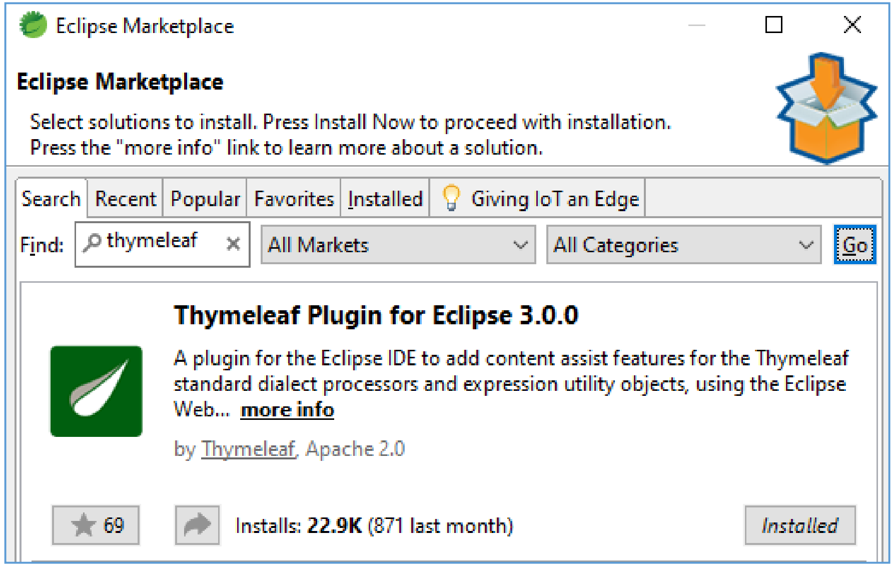
1. Install Thymeleaf plugin in STS using ‘Help > Eclipse Marketplace…’ or site URL *http://www.thymeleaf.org/eclipse-plugin-update-site/*
2. Add Thymeleaf dependency in pom.xml
3. Add ‘Thymeleaf nature’ to your project & restart STS.
4. Start using thymeleaf API support in spring boot development.

Note: Refer to *https://github.com/thymeleaf/thymeleaf-extras-eclipse-plugin*

Thymeleaf plugin in STS

Click on ‘Help > Eclipse Marketplace…’ option.

Search for ‘Thymeleaf’. You will get below plugin, just click on ‘Install’:



Eclipse Marketplace

Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.

Find: thymeleaf

Thymeleaf Plugin for Eclipse 3.0.0

A plugin for the Eclipse IDE to add content assist features for the Thymeleaf standard dialect processors and expression utility objects, using the Eclipse Web... **more info**

by Thymeleaf, Apache 2.0

69 Installs: **22.9K** (871 last month) Installed

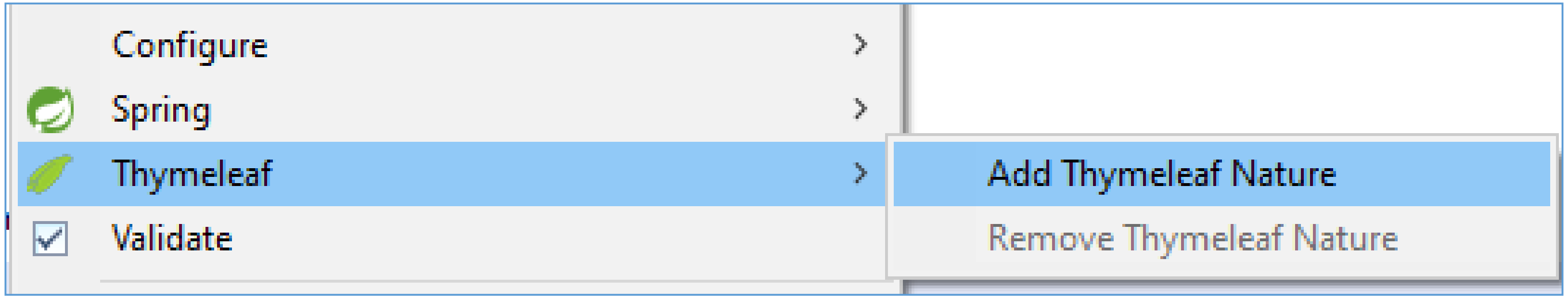
Thymeleaf maven dependency

Below is the thymeleaf dependency to be added in pom.xml

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

Thymeleaf nature of Spring Boot App

In order to take advantage of Thymeleaf plugin & to make content assist available to *all* of the HTML files in your project, you need to right click on the project & choose 'Thymeleaf nature' as shown below:



Running first Thymeleaf page

- Create a Spring Boot app with Thymeleaf maven dependencies added.
- Write a HTML page inside \resources\templates folder with thymeleaf custom tags.
- Write a controller that returns this HTML page as response.

Writing MVC Controller in Spring Boot

```
@Controller
public class SampleController {
  @RequestMapping(value="contactus")
  public String contactUs(Model model) {
        model.addAttribute("fname", "Anand");
        return "contact-us";
  }
}
```

Writing Thymeleaf template

```
//contact-us.html

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

  <body>

        <h1 th:text="${fname}"></h1>

  </body>

</html>
```

Standard Expression Syntax

Most Thymeleaf attributes allow their values to be set as or containing expressions, which we will call Standard Expressions. We have 5 types of Standard Expressions supported by Thymeleaf.

Sr. No.	Standard Expression Type	Syntax
1	Variable expressions	${...}
2	Selection expressions	*{...}
3	Message (i18N) expressions	#{...}
4	Link (URL) expressions	@{...}
5	Fragment expressions	~{...}

Standard Expression Syntax

- **Variable expressions:**

  Variable expressions are used to retrieve data from Model attributes.

  ```
  <span th:text="${user.address.city}">
  ```

- **Selection expressions:**

  Selection expressions are just like variable expressions, except they will be executed on a previously selected object instead of the whole context variables map.

  ```
  <div th:object="${user}">
    <span th:text="*{firstName}">...</span>
  </div>
  ```

Standard Expression Syntax

- **Message i18N expressions:**

  Message expressions are used to retrieve data from external sources(.properties file). It is useful in Internationalization(i18N).

  <span th:text=“#{login.success.msg}">

- **Link (URL) expressions:**

  Link expressions are used to specify the URL or location of css or js files.

  <a th:href="@{/order/list}">...</a>

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

  <script type="text/javascript" th:src="@{/js/main.js}" ></script>

Standard Expression Syntax

- **Fragment expressions:**

  Fragment expressions are an easy way to represent fragments of markup and move them around templates.

```
//general.html

<footer th:fragment="footer">

//mypage.html

<div th:replace="fragments/general.html :: footer"></div>
```

Adding CSS & JS files in Thymeleaf app

- Custom CSS & JS files are added inside resources\static folder.
- Generally we create separate sub folders like static\css & static\js to place our CSS & JS files.
- In order to access CSS & JS files from Thymeleaf template page, use below syntax:

```
<link th:href="@{/css/main.css}" />

<script type="text/javascript" th:src="@{/js/main.js}" />
```

Conditional statements & loops in Thymeleaf template

- Conditional statements allows us to render an HTML element depending on a provided condition.
- We use <th:if> & <th:unless> tags to handle a condition.
- Also, we use <th:each> tag to iterate over a collection.

```
<ul th:each="user:${users}">
    <li th:if="${user.location}=='Washington'" th:text="${user}"></li>
    <li th:unless="${user.location}=='Washington'" th:text="${user}"></li>
</ul>
```

Conditional statements & loops in Thymeleaf template

- Instead of adding multiple conditional statements using <th:if>, we can also use switch case using <th:switch> tag.

```
<div th:each="user:${users}">
    <div th:switch="${user.location}">
        <p th:case="'New York'">New York location</p>
        <p th:case="'Washington'">Washington location</p>
        <p th:case="'London'">London location</p>
        <p th:case="*">Location not found</p>
    </div>
</div>
```

Creating local variables using th:with

Thymeleaf template allows us to create local variables inside the template using th:with tag.

```
<div th:with="firstArticle=${articles[0]}">
    <a th:text="${firstArticle.name}" th:href="${firstArticle.url}"></a>
</div>
```

Thank you!!