Spring MVC

Prof. Luiz Real

O que é Thymeleaf?

- Thymeleaf é uma engine de template para Java que permite criar páginas HTML, XML, JavaScript, CSS, etc., com conteúdo dinâmico.
- Ao contrário de outras engines que exigem uma etapa separada de processamento para renderizar o conteúdo, o Thymeleaf é capaz de processar os templates tanto no servidor (como JSP) quanto no cliente (no navegador, com um processamento mínimo), tornando o desenvolvimento mais eficiente e facilitando o teste dos templates localmente.
- Isso significa que você pode ver uma prévia do seu HTML já com os dados inseridos, diretamente no navegador, sem a necessidade de um servidor rodando.

Recursos Principais do Thymeleaf

- **Integração com Spring:** Se integra perfeitamente com o Spring Framework, facilitando a inserção de dados do modelo (model) nos templates.
- **Sintaxe Natural:** Utiliza uma sintaxe natural que se parece muito com HTML puro, tornando os templates mais fáceis de ler e manter. Ele evita a necessidade de tags específicas ou códigos muito complexos dentro do HTML.
- **Processamento no Servidor e no Cliente:** Como já mencionado, pode processar templates tanto no servidor quanto no cliente (em modo de processamento mínimo), o que simplifica o teste e o desenvolvimento.
- **Expressões de Template:** Permite o uso de expressões para manipular dados, realizar cálculos e lógica condicional diretamente nos templates.
- **Dialects:** Oferece suporte a dialects (dialetos), que extendem sua funcionalidade, permitindo a integração com outros frameworks e bibliotecas.

Exemplo de Código com Spring Boot

Vamos criar um simples exemplo de um controller Spring Boot que retorna uma página HTML processada pelo Thymeleaf.

Dependência no Gradle:

implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'

Vamos criar um controller:

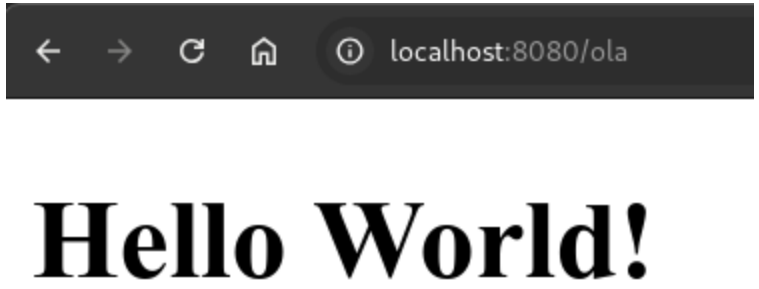
```java
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller new *
public class HelloController {
    @GetMapping("/ola") new *
    public String ola(Model model) {
        model.addAttribute( attributeName: "message", attributeValue: "Hello World!");
        return "hello"; // Nome do template HTML
    }
}
```

E um template HTML:

```html
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Hello World</title>
</head>
<body>
    <h1><span th:text="${message}"></span></h1>
</body>
</html>
```

Exemplo de Código com Spring Boot



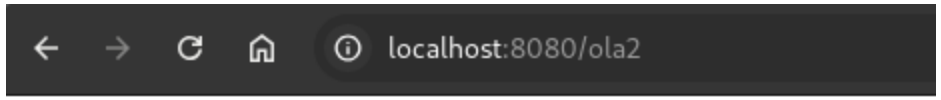
Observe o namespace xmlns:th="http://www.thymeleaf.org" no HTML. Ele indica ao Thymeleaf que ele deve processar as tags com **th:**

th:text="${nome}" substitui o conteúdo da span pelo valor da variável **message** passada no modelo.

O objeto **Model** é usado para passar dados para a template.

Ao executar o Spring Boot e acessar localhost:8080/ola no navegador, você verá "Hello, World!".

Exemplo de Código com Spring Boot

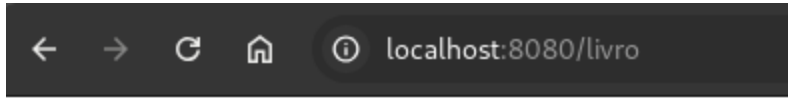


Hellooooo World!

ModelAndView é uma classe que combina um modelo de dados com o nome de uma view (template). Ele oferece mais controle sobre o processo de renderização, permitindo definir parâmetros adicionais para a view.

```java
@GetMapping("/ola2")  new *
public ModelAndView ola2() {
    ModelAndView mv = new ModelAndView( viewName: "hello");
    mv.addObject( attributeName: "message",  attributeValue: "Hellooooo World!");
    return mv;
}
```

Exemplo de Código com Spring Boot



Odisseia

Um livro de: Homero

Você também pode passar **objetos** no modelo:

```
@GetMapping("/livro")  new *
public String ola(Model model) {
    Livro livro = new Livro();
    livro.setNome("Odisseia");
    livro.setAutoria("Homero");
    model.addAttribute( attributeName: "livro", livro);
    return "livro"; // Nome do template HTML
}
```

```
<body>
    <h1><span th:text="${livro.getNome()}"></span></h1>
    <p>Um livro de: <span th:text="${livro.getAutoria()}"></span></p>
</body>
```

Criando um template com fragments

header

```html
<!DOCTYPE html>
<html lang="pt-BR" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Cabeçalho</title>
</head>
<body>
    <div th:fragment="header">
        <h1>Conteúdo do Cabeçalho</h1>
        <hr />
    </div>
</body>
</html>
```

Criando um template com fragments

footer

```html
<!DOCTYPE html>
<html lang="pt-BR" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Rodapé</title>
</head>
<body>
    <div th:fragment="footer">
        <hr />
        <h2>Conteúdo do Rodapé</h2>
    </div>
</body>
</html>
```

Criando um template com fragments

template

```
<!DOCTYPE html>
<html lang="pt-BR" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Template</title>
</head>
<body>
    <div th:replace="~{fragments/header :: header}"></div>
    <div>Conteúdo</div>
    <div th:replace="~{fragments/footer :: footer}"></div>
</body>
</html>
```

Criando um template com fragments

```
@GetMapping("/template")
public String template() {
    return "templateVazio";
}
```

Dialects

Como vimos, o **Thymeleaf** é um motor de template altamente extensível que permite definir conjuntos personalizados de atributos ou tags em templates, avaliando **expressões** e aplicando **lógicas específicas** conforme necessário.

Por padrão, ele oferece **dialetos** padrão chamados *Standard* e *SpringStandard*, que fornecem um conjunto de funcionalidades adequadas para a maioria dos cenários.

Esses dialetos são identificados por atributos com o prefixo **`th`**, como:

`<span th:text="...">`

A principal diferença entre eles é que o *SpringStandard* inclui recursos específicos para integração com aplicações **Spring MVC**, utilizando o **Spring Expression Language** (SpEL*) em vez do OGNL (Object-Graph Navigation Language) para avaliação de expressões.

*SpEL permite acessar beans do Spring e algumas classes java que o OGNL não acessa.

```
<div th:text="${T(java.lang.Math).sqrt(user.age)}"></div>
<div th:text="${userService.findById(1).name}"></div>
```

Sintaxe de Expressões Padrão no Thymeleaf

Os atributos do Thymeleaf permitem o uso de expressões em seus valores, conhecidas como Expressões Padrão *(Standard Expressions)*. Elas são classificadas em cinco tipos:

${...} : Variable expressions

***{...}** : Selection expressions

#{...} : Message (i18n) expressions

@{...} : Link (URL) expressions

~{...} : Fragment expressions

Sintaxe de Expressões Padrão no Thymeleaf

Variable Expressions (**${...}**):

- Avaliam variáveis do contexto (ou model attributes no Spring)
- Você as encontra como valores de atributo:

```
<span th:text="${book.author.name}">
```

- Essa expressão equivale a:

```
((Book)context.getVariable("book")).getAuthor().getName()
```

- Mas podemos encontrar expressões variáveis em cenários que não envolvem apenas a saída, mas um processamento mais complexo, como condicionais, iteração...

```
<li th:each="book : ${books}">
```

Sintaxe de Expressões Padrão no Thymeleaf

Selection Expressions (***{...}**):

- Semelhantes às expressões de variável, mas operam sobre um objeto previamente selecionado:

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

Sintaxe de Expressões Padrão no Thymeleaf

Message Expressions (**#{...}**):

- Recuperam mensagens específicas de locale a partir de fontes externas (.properties), facilitando a internacionalização:

`#{main.title}`

- busca a mensagem com a chave **`main.title`**

Sintaxe de Expressões Padrão no Thymeleaf

Link Expressions (**@{...}**):

- Constroem URLs, adicionando informações de contexto e sessão conforme necessário:

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

- gera um link para **`/order/list`**, considerando o contexto da aplicação

Sintaxe de Expressões Padrão no Thymeleaf

Fragment Expressions (**~{...}**):

- Referenciam fragmentos de templates, permitindo a inclusão de partes reutilizáveis:

```
<div th:insert="~{fragments/header :: menu}">
```

- insere o fragmento **menu** do template **header**

Sintaxe de Expressões Padrão no Thymeleaf

- Um exemplo ilustrativo de como essas expressões podem ser utilizadas em um template Thymeleaf:

```
<div th:object="${book}">
  <h1 th:text="*{title}">Título do Livro</h1>
  <p th:text="*{author.name}">Nome do Autor</p>
  <p th:text="#{book.description}">Descrição do Livro</p>
  <a th:href="@{/books/edit/{id}(id=*{id})}">Editar Livro</a>
</div>
```

Sintaxe de Expressões Padrão no Thymeleaf

Literais e Operações

Literals:

Text literals: 'one text', 'Another one!',...

Number literals: 0, 34, 3.0, 12.3,...

Boolean literals: true, false

Null literal: null

Literal tokens: one, sometext, main,...

Text operations:

String concatenation: +

Literal substitutions: |The name is ${name}|

Arithmetic operations:

Binary operators: +, -, *, /, %

Minus sign (unary operator): -

Sintaxe de Expressões Padrão no Thymeleaf

Literais e Operações

Boolean operations:

Binary operators: and, or

Boolean negation (unary operator): !, not

Comparisons and equality:

Comparators: >, <, >=, <= (gt, lt, ge, le)

Equality operators: ==, != (eq, ne)

Conditional operators:

If-then: (if) ? (then)

If-then-else: (if) ? (then) : (else)

Default: (value) ?: (defaultvalue)