Thymeleaf

Certified Developer
The Ultimate Tech Degree

DigitalHouse >
Coding School

Índice

1 | Sintaxes

Sintaxes

A sintaxe do template Thymeleaf é definida nas páginas HTML pelo prefixo **th**. Os navegadores irão ignorar os nomes de elementos e atributos que não compreende, ou seja, o **th**, de modo que a página permanecerá válida.

O atributo **xmlns** define o namespace para th.
Isso permite definir nomes de atributos e elementos exclusivos em um documento XML (ou HTML).

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Login Page</title>
</head>
<body>
      <h1>Digite seu nome:</h1>
</body>
</html>
```

th:text

Para exibir o texto no template, usamos o atributo **th:text**.

```
<body>
    <p th:text="'Obrigado, ' + ${userName} + ' por praticar
    - Spring Boot!!'" />
    <a href="/">Home</a>
</body>
```

th:if

Para mostrar ou ocultar elementos HTML, usamos o atributo **th:if**. Por exemplo, se queremos ocultar ou exibir uma parte do nosso HTML, devemos colocar a seguinte anotação no elemento (tag): **th:if="${condition}"**, e se a condição for verdadeira, o elemento será mostrado. Caso contrário, não será mostrado.

```
<body>
    <p th:if="${isVisible}">This is visible!</p>
    <a href="/">Home</a>
</body>
```

Se a variável ***isVisible*** é verdadeira (=true), a tag **p** será visualizada.

th:class

Permite que você modifique as classes de um elemento dinamicamente em tempo de execução. Por exemplo, se quisermos modificar o estilo da tag **p** dependendo se a variável **condition** é verdadeira ou falsa, podemos escrever a seguinte linha de código:

```
<p th:class="${condition} ? 'white' : 'black'"></p>
```

Se a variável condition for verdadeira, quando o HTML for renderizado, teremos:

```
<p class="white"></p>
```

Se a variável condition for false, quando o HTML for renderizado, teremos:

```
<p class="black"></p>
```

th:classappend

Permite adicionar estilos às classes de um elemento dinamicamente em tempo de execução. Por exemplo, se quisermos adicionar mais estilos à tag **p** dependendo se a variável **condition** é verdadeira ou falsa, podemos escrever a seguinte linha de código:

```
<p class="panel" th:classappend="${condition} ? 'white' :
'black'"></p>
```

Se a variável condition for verdadeira, quando o HTML for renderizado, teremos:

```
<p class="panel white"></p>
```

Se a variável condition for false, quando o HTML for renderizado, teremos:

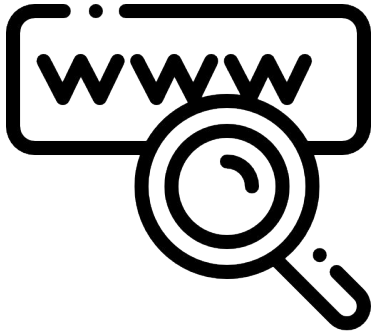
```
<p class="panel black"></p>
```

2 | URLs

th:href

Para construir URLs devemos ter em mente se são URLs **absolutas ou relativas**.

Vamos ver como elas se diferem.



URLs absolutas

Elas são utilizadas para criar links que apontam para outros servidores.

Começando com **http://** ou **https://**, a menos que você tenha um filtro de reescrita de URL em seu servidor, o moto Thymeleaf não os alterará.

Por exemplo, se tivermos que adicionar a URL: https://www.digitalhouse.com/

```
<a th:href="@{https://digitalhouse.com}">digital house</a>
```

O resultado será o seguinte:

```
<a href="https://digitalhouse.com">digital house</a>
```

URLs relativas

Esse tipo de URL são as **mais utilizadas em aplicações web**. As URLs relativas podem ser relativas ao caminho atual ou à raiz da aplicação. Sendo relativas à raíz quando começam com “/”.

Por exemplo, um link direcionando para uma página de nosso projeto “myapp”.

```
<a th:href="@{dh}’’>digital house</a>
```

O resultado será o seguinte:

```
<a href="/myapp/dh">digital house</a>
```

th:each

É utilizado quando precisamos percorrer (iterar) coleções de diferentes tipos de objetos.

Existem diversos objetos que o Thymeleaf nos permite percorrer:

- Objetos que implementam a interface **java.util.Iterable**;
- Objetos que implementam a interface **java.util.Enumeration**;
- Objetos que implementam a interface **java.util.Iterator**;
- Objetos que implementam a interface **java.util.Map**;
- E matrizes (arrays).

Vamos ver como podemos iterar/percorrer uma lista de clientes.

```
public class Cliente {

    private String firstName;
    private String lastName;
    private String email;
    private int age;
}
@Controller
public class clienteController {
    @Autowired
    private ClienteService clienteService;

    @GetMapping("/clientes")
    public String obterClientes(Model model) {

        model.addAttribute("clientes",
            clienteService.getClientes());

        return "vistaCliente";
    }
}
```

No atributo **clientes,** enviamos a lista de clientes que obtivemos do serviço clienteService.
Em seguida, iteramos os clientes para que nossa view mostre cada um dos clientes retornados pelo serviço clienteService.

```
<!DOCTYPE HTML>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"/>
    <title>Clientes</title>
</head>
<body>
<h1>Lista de Clientes</h1>
<table>
    <tr>
        <th>No</th>
        <th>First Name</th>
        <th>Last Name</th>
        <th>Age</th>
        <th>Email</th>
    </tr>
    <tr th:each="cliente, custStatus : ${clientes}">
        <td th:text="${custStatus.count}"></td>
        <td th:text="${cliente.firstName}"></td>
        <td th:text="${cliente.lastName}"></td>
        <td th:text="${cliente.age}"></td>
        <td th:text="${cliente.email}"></td>
    </tr>
</table>
</body>
</html>
```

Vamos ver como ficaria o template **CustomerView.html**.

O motor Thymeleaf repetirá o bloco do elemento que contém o atributo **th:each** com cada item existente na coleção.


DigitalHouse>
Coding School