Spring Essentials

Spring Fundamentals

SoftUni Team

Technical Trainers

Software University

https://softuni.bg

Table of Content

Questions

sli.do

#java-web

Thymeleaf

The Templating Engine

What is Thymeleaf?

- Thymeleaf is a modern server-side Java **template engine** used in Spring
- It allows us to
  - Use variables in our views
  - Execute operations on our variables
  - Iterate over collections
  - Make our views dynamical

How to Use Thymeleaf?

- Use Spring Initializer to import Thymeleaf, or use a dependency

```
In Maven:
```

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

```
In Gradle:
```

```
dependencies {
       compile("org.springframework.boot:spring-boot-starter-thymeleaf")
}
```

- Define the Thymeleaf library in your html file

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

Thymeleaf Tags and Attributes

- All Thymeleaf tags and attributes begin with **th:** by default
- Example of Thymeleaf attribute

```
<p th:text="${user.name}">Some text</p>
```

- Example of Thymeleaf tag(element processor)

```
<th:block>
  ...
</th:block>
```

- **th:block** is an attribute container that **disappears** in the HTML

Thymeleaf Standard Expressions

- Variable Expressions

```
${...}
```

- Selection Expressions

```
*{...}
```

- Accessing Bean

```
${@...}
```

- Link (URL) Expressions

```
@{...}
```

- Fragment Expressions

```
~{...}
```

Thymeleaf Variable Expressions

- Variable Expressions are executed on the context variables

```
${...}
```

- Examples

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

```
${title}
```

```
${game.id}
```

If else & switch

- **If – else**

```
<div th:if="${student.passExam}">Show results</div>
<div th:unless="${student.passExam}">Not pass</div>
```

- **Switch**

```
<div th:switch="${user.role}">
  <p th:case="'admin'">User is an administrator</p>
  <p th:case="#{roles.manager}">User is a manager</p>
</div>
```

Default expressions (Elvis operator)

- A special kind of conditional value **without a 'then' part**. It is equivalent to the **Elvis** operator present in some languages

```
<p>Age:
   <span th:text="*{age} ?: 'missing age'"> </span>
</p>
```

- Equivalent to:

```
<p>Age:
   <span th:text="*{age != null}? *{age}:'missing age'"></span>
</p>
```

Thymeleaf Link Expressions

- Link Expressions are used to build URLs

```
@{...}
```

- Example

```
<a th:href="@{/register}">Register</a>
```

- You can also pass query string parameters

```
<a th:href="@{/details(id=${game.id})}">Details</a>
```

```
Result -> /details?id=3
```

- Create dynamic URLs

```
<a th:href="@{/games/{id}/edit(id=${game.id})}">Edit</a>
```

```
Result -> /games/3/edit
```

Iteration

- When we want to **iterate** over collection

```
<tr th:each="s : ${students}">
        <td th:text="${s.name}"></td>
        <td th:text="${s.score}"></td>
        <td th:text="${s.age}"></td>
</tr>
```

- We can attach the **object** to the parent element

```
<tr th:each="s : ${students}" th:object="${s}">
        <td th:text="*{name}"></td>
        <td th:text="*{score}"></td>
        <td th:text="*{age}"></td>
</tr>
```

Appending and prepending

- **th:attrappend** and **th:attrprepend** attributes, which append (suffix) or prepend (prefix) the result of their evaluation to the existing attribute values

```
<input type="button" value="Play"
         class="btn" th:attrappend="class=${' ' + cssStyle}" />
```

- **th:classappend**:

```
 <li th:classappend="${module == 'home' ? 'active' : ''}">
```

Forms in Thymeleaf

- In Thymeleaf you can create almost normal HTML forms

```
<form th:action="@{/users}"th:method="post">
    <input type="number" name="id"/>
    <input type="text" name="name"/>
    <button type="submit"/>
</form>
```

- You can have a controller that will accept an object of given type

```
@PostMapping("/user")
public ModelAndView register(User user) { ... }
```

Fragments in Thymeleaf

- Often we want to include in our templates **fragments** from other **templates**
  - Common uses for this are footers, headers, menus
  - Define the fragments available for inclusion, which we can do by using the **th:fragment** attribute
  - After than we can easily include in our home page using one of the **th:include** or **th:replace** attributes

Fragments in Thymeleaf

- Create class with fragments

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
  <body>
    <div th:fragment="copy">
      &copy; Spring Team 2021
    </div>
  </body>
</html>
```

Fragments in Thymeleaf

- Easily include in our home page using one of the **th:include** or **th:replace** attributes

```
<body>
  ...
  <footer th:include="footer::copy"></footer>
        //OR
  <footer th:replace="footer::copy"></footer>
  ...
</body>
```

Difference between include and replace

```
<footer th:include="footer :: copy"></footer>
<footer th:replace="footer :: copy"></ footer>
 ...
```

- The result is

```
<footer>
    &copy; Spring Team 2021
</footer>
<div>
    &copy; Spring Team 2021
</div>
...
```

Fragments in Thymeleaf

- Create Fragment **without th:fragment**

footer.html

```
<th:block>
  <footer> Spring Team 2020 </footer>
</th:block>
```

- Use Fragment

index.html

```
...
<th:block th:include="~{/fragments/footer}"></th:block>
...
```

Additional Spring Functionalities

ModelAttribute (1)

- When the annotation is used at the **method level**, it indicates **the purpose of that method**
  - to add one or more model attributes
- In the example, a method adds an attribute named message to all models defined in the controller class

```
@ModelAttribute
public void addAttributes(Model model) {
       model.addAttribute("message", "Welcome to SoftUni!");
}
```

ModelAttribute (2)

- When used as a **method argument**, it **indicates the argument** should be retrieved from the model
- When **not present**, it should be **first instantiated** and then added to the model.
- Once **present in the model**, the arguments **fields should be populated** from all request parameters that have matching names.

ModelAttribute Examples

- Example of using **@ModelAttribute** as a method argument

```
@RequestMapping(value = "/cars/add",
       method = RequestMethod.POST)
public String submit(@ModelAttribute("car")Car car) {
// Some code ...
 return "carView";
}
```

@CrossOrigin

- **@CrossOrigin**
  - marks the annotated method or type as permitting cross origin requests

```
@CrossOrigin(origins = "http://example.com")
@RequestMapping("/hello")
public String hello() {
       return "Hello World!";
}
```

@Qualifier (1)

- We use **@Qualifier** along with **@Autowired** to provide the bean id or bean name

```
@Component
@Qualifier("bike")
class Bike implements
Vehicle {
   private String make;
   private String model;
}
```

```
@Component
@Qualifier("car")
class Car implements
Vehicle {
   private String make;
   private String model;
   private Integer seats;
}
```

@Qualifier (2)

- If we want to get Bike, we need to specify it with adding **@Qualifier("bike")** before injecting Vehicle

```
@Autowired
Biker(@Qualifier("bike") Vehicle vehicle) {
        this.vehicle = vehicle;
}
```

@Primary (1)

- We can use **@Primary** to simplify this case:
  - if we mark the most frequently used bean with **@Primary**

```
@Component
@Primary
class Car implements
Vehicle {...}
```

```
@Component
class Bike implements
Vehicle {}
```

@Primary (2)

- The example of **@Primary** use case

```
@Component
class Driver {
        @Autowired
        Vehicle vehicle;
}
```

```
@Component
class Biker {
        @Autowired
        @Qualifier("bike")
        Vehicle vehicle;
}
```

Working with Http Sessions, Cookies and Headers

Working with the Session

- The session will be **injected from the IoC** container when called

```
@GetMapping("/")
public String home(HttpSession httpSession) {
  …
  httpSession.setAttribute("id", 2);
  …
}
```

- Later the session attributes can be accessed from Thymeleaf using the expression syntax and the **#session** object

Reading HTTP Cookie

- The annotation **@CookieValue**

```
@GetMapping("/")
public String readCookie(@CookieValue(value = "username",
defaultValue = "Guest") String username) {
       return "login";
}
```

Setting HTTP Cookie (1)

- Using the **ResponseCookie** object

```
ResponseCookie cookie = ResponseCookie.from("username", "pesho")
     .httpOnly(true)
     .secure(true)
     .path("/")
     .maxAge(60)
     .domain("softuni.bg")
     .build();
ResponseEntity
     .ok()
     .header(HttpHeaders.SET_COOKIE, cookie.toString())
     .build();
```

Setting HTTP Cookie (2)

- @CookieValue

```
@GetMapping("/change-username")
public String setCookie(HttpServletResponse response) {
       // create a cookie
       Cookie cookie = new Cookie("username", "Pesho");
       //add cookie to response
       response.addCookie(cookie);
       return "index";
}
```

RequestHeader

- Reading **HTTP Header**

```
@GetMapping("/greeting")
public ResponseEntity<String> greeting(
@RequestHeader("accept-language") String language) {
       // code that uses the language variable
       return new ResponseEntity<String>("greeting",
       HttpStatus.OK);
}
```

ResponseStatus

- We can specify the desired **HTTP status** of the response

```
@RequestMapping(method = RequestMethod.POST)
@ResponseStatus(HttpStatus.CREATED)
public void storeEmployee(@RequestBody Employee employee) {
    ...
}
```

Request & Response Body

@RequestBody (1)

- Maps the **HttpRequest body** to a transfer or domain object, enabling automatic deserialization of the inbound HttpRequest body on to a Java objects

```
@PostMapping("/students/add")
public ResponseEntity postController(
@RequestBody StudentAddBindingModel bindingModel){
       myService.add(bindingModel);
       return ResponseEntity.ok(HttpStatus.OK);
}
```

@ResponseBody (2)

- Tells a controller that the object returned is automatically serialized into JSON and passed back into the **HttpResponse object**

```
@GetMapping("/response")
@ResponseBody
public Exercise getLastEx() {
       // Get exercise from service
       return exercise;
}
```

```
{"id":"0b5963eb-4f4d-4718-bd34-
d0206d80046a","name":"SPRING DATA
INTRO","startedOn":"2021-01-
14T19:26:00","dueDate":"2021-02-05T19:26:00"}
```

Summary

- Thymeleaf
  - Work with variables and objects
  - Create forms
- HTTP Sessions
  - Cookies
  - Headers
- Additional Spring Extras and Components

Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

SoftUni Diamond Partners

Software University

DRAFT KINGS

INDEAVR
Serving the high achievers

Coca-Cola HBC Bulgaria

SCHWARZ

LIDL

Kaufland

SCHWARZ Produktion

pre zero

Postbank
Решения за твоето утре

POKERSTARS | CAREERS

SUPER HOSTING .BG

SOFTWARE GROUP

SmartIT

PHAR VISION

createX

taulia®

Educational Partners

Software University

CODE IT UP

VIRTUAL RACING SCHOOL

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License