

Java MVC Frameworks

Spring Essentials. Thymeleaf and Controllers



SoftUni Team
Technical Trainers



SoftUni
Foundation



Software University

<http://softuni.bg>

1. Thymeleaf

- The templating engine

2. Spring Controllers

- Annotations
- Http Mappings
- Redirect Attributes



sli.do

#java-web

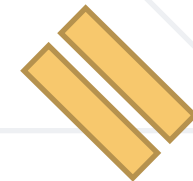


Thymeleaf

The Templating Engine

What is Thymeleaf?

- Thymeleaf is a view 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 Initializr to import Thymeleaf, or use this dependency in your **pom.xml**:

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

- Define the Thymeleaf library in your html file:

```
<html xmlns="http://www.w3.org/1999/xhtml"  
      xmlns:th="http://www.thymeleaf.org">
```

Change Thymeleaf Version

- You also need to change the Thymeleaf version in your **pom.xml**:

```
<properties>  
  <thymeleaf.version>  
    3.0.9.RELEASE  
  </thymeleaf.version>  
  <thymeleaf-layout-dialect.version>  
    2.2.0  
  </thymeleaf-layout-dialect.version>  
</properties>
```

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

```
<p th:text="Example"></p>
```

- Example of Thymeleaf tag(element processor):

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

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

- Variable Expressions

`${...}`

- Selection Expressions

`*{...}`

- Link (URL) Expressions

`@{...}`

- Fragment Expressions

`~{...}`

- Variable Expressions are executed on the context variables

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

- Examples:

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

```
${title}
```

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

- 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>
```

- Create dynamic URLs

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

- In Thymeleaf you can create almost normal HTML forms:

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

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

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



Spring Controllers

Annotations, IoC Container

- Defined with the `@Controller` annotation:

```
@Controller  
public class HomeController {  
    ...  
}
```

- Controllers can contain multiple actions on different routes.

- Annotated with with `@RequestMapping(...)`

```
@RequestMapping("/home")  
public String home() {  
    return "home-view";  
}
```

- Or

```
@RequestMapping("/home")  
public ModelAndView home(ModelAndView mav) {  
    mav.setViewName("home-view");  
    return mav;  
}
```

- Problem when using `@RequestMapping` is that it accepts all types of request methods (get, post, put, delete, head, patch...)
- Execute only on GET requests :

```
@RequestMapping(value="/home", method=RequestMethod.GET)  
public String home() {  
    return "home-view";  
}
```


- Easier way to create route for a GET request:

```
@GetMapping("/home")  
public String home() {  
    return "home-view";  
}
```

- This is alias for **RequestMapping** with method GET

- Similar to the **GetMapping** there is also an alias for **PostMapping** with method POST:

```
@PostMapping("/register")  
public String register() {  
    ...  
}
```

- Similar annotations exist for all other types of request methods

- Passing a string to the view:

```
@GetMapping("/")  
public String welcome(Model model) {  
    model.addAttribute("name", "Pesho");  
  
    return "welcome";  
}
```

- The **Model** object will be automatically passed to the view as context variables and the attributes can be accessed from Thymeleaf using the **Expression syntax** - `${name}`

- 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:

```
${session.id}
```

- Getting a parameter from the query string:

```
@GetMapping("/details")  
public String details(@RequestParam("id") Long id) {  
    ...  
}
```

- `@RequestParam` can also be used to get POST parameters

```
@PostMapping("/register")  
public String register(@RequestParam("name") String name)  
{ ... }
```

- Getting a parameter from the query string:

```
@GetMapping("/comment")  
public String comment(@RequestParam(name="author",  
defaultValue = "Anonymous") String author)  
{ ... }
```

- Making parameter optional:

```
@GetMapping("/search")  
public String search(@RequestParam(name="sort", required  
= false) String sort)  
{ ... }
```

- Spring will automatically try to fill objects with a form data

```
@PostMapping("/register")  
public String register(@ModelAttribute UserDTO userDto) {  
    ...  
}
```

- The input field names must be the same as the object field names

- Redirecting after POST request:

```
@PostMapping("/register")  
public String register(@ModelAttribute UserDTO userDto) {  
    ...  
    return "redirect:/login";  
}
```


- Redirecting with query string parameters

```
@PostMapping("/register")  
public String register(@ModelAttribute UserDTO userDto,  
    RedirectAttributes redirectAttributes) {  
    redirectAttributes.addAttribute("errorId", 3);  
    return "redirect:/login";  
}
```

- Keeping objects after redirect

```
@PostMapping("/register")  
public String register(@ModelAttribute UserDTO userDto,  
    RedirectAttributes redirectAttributes) {  
    ...  
    redirectAttributes.addFlashAttribute("userDto",  
userDto);  
  
    return "redirect:/register";  
}
```

- Thymeleaf is a powerful view engine
 - You can work with variables and helper Objects
 - You can easily create forms
- The Spring Controllers:
 - You can create routings on actions and controllers
 - You have access to the `HttpRequest`, `HttpResponse`, `HttpSession` and others
 - You can redirect between actions



Questions?



SoftUni



**Software
University**



**SoftUni
Svetlina**



**SoftUni
Creative**



**SoftUni
Digital**



**SoftUni
Foundation**



**SoftUni
Kids**

SoftUni Diamond Partners



XSsoftware



SBTech
we know sports



telenor



SoftwareGroup
doing it right

NETPEAK



SmartIT



Postbank

Решения за твоето утре

**SUPER
HOSTING
.BG**

INDEAVR

Serving the high achievers



INFRAGISTICS®

LIEBHERR



aeternity



SoftUni Organizational Partners



OneBit
SOFTWARE



WORLD
OF
MYTHS

- Software University – High-Quality Education and Employment Opportunities
 - softuni.bg
- Software University Foundation
 - <http://softuni.foundation/>
- Software University @ Facebook
 - facebook.com/SoftwareUniversity
- Software University Forums
 - forum.softuni.bg



- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license

