FREE eBook

LEARNING thymeleaf

#thymeleaf

Table of Contents

About 1

Chapter 1: Getting started with thymeleaf 2

- Remarks 2
- Versions 2
- Examples 2
  - Configuration 2
  - Using checkboxes 3
  - Ajax form submition with Jquery 4
  - Replacing fragments with ajax 5
  - Form Submission 7

Chapter 2: Expression Utility Objects 9

- Examples 9
  - Format date 9
  - String length 9
  - String contains 9
  - Parsing date 9
  - Format decimal 10

Chapter 3: Externalizing Text in Thymeleaf 11

- Examples 11
  - Localization 11
  - Localization messages with parameters 11

Chapter 4: Spring Security and Thymeleaf 13

- Examples 13
  - Secure your WebApp with Login and Logout 13

Maven Dependencies 13

Create your WebApp 13

Secure your WebApp 14

Create the login page 14

Access user properties 15

- Displaying something for authenticated users only 15

Display username.....16
Display different content to different roles.....16
Chapter 5: Using Lists with Thymeleaf.....17
Examples.....17
Using list in select.....17
Form table.....17
Credits.....18

About

Chapter 1: Getting started with thymeleaf

Remarks

Thymeleaf is a template engine, a library written in JAVA. It allows a developer to define a HTML, XHTML or HTML5 page template and later fill it with data to generate final page. Therefore it realizes a *Model-View* part of a Model-View-Controller pattern.

Thymeleaf's important design principle is that a template itself has to be properly written (X)HTML.

Versions

Version	Date	Latest Release	Date
3.x.x	2016-05-08	3.0.6	2017-05-07
2.x.x	2012-02-09	2.1.5	2016-07-11

Examples

Configuration

To get started with Thymeleaf visit official download page.

Maven dependency

```
<dependency>
  <groupId>org.thymeleaf</groupId>
  <artifactId>thymeleaf</artifactId>
  <version>3.0.1.RELEASE</version>
</dependency>
```

Gradle dependency

```
compile group: 'org.thymeleaf', name: 'thymeleaf', version: '3.0.1.RELEASE'
```

Example configuration

Starting from version 3.0, Thymeleaf supports only Java config.

```
public ViewResolver viewResolver() {
    ThymeleafViewResolver resolver = new ThymeleafViewResolver();
    resolver.setTemplateEngine(templateEngine());
    resolver.setCharacterEncoding("UTF-8");
    resolver.setContentType("text/html; charset=UTF-8");
    return resolver;
}
```

In `viewResolver()` method you can setup e.g. encoding and content type for views. more information

```
public TemplateEngine templateEngine() {
    SpringTemplateEngine engine = new SpringTemplateEngine();
    engine.setTemplateResolver(templateResolver());
    return engine;
}
```

In `templateEngine()`, you can add custom dialects. For example to add Spring Security dialect you can do this like this `engine.addDialect(new SpringSecurityDialect());`

```
public ITemplateResolver templateResolver() {
    SpringResourceTemplateResolver resolver = new SpringResourceTemplateResolver();
    resolver.setApplicationContext(applicationContext);
    resolver.setPrefix("/views/");
    resolver.setSuffix(".html");
    resolver.setTemplateMode(TemplateMode.HTML);
    resolver.setCharacterEncoding("UTF-8");
    return resolver;
}
```

Look at setter for prefix and suffix in `templateResolver()` method. It tells Thymeleaf that, every time controller will return view, Thymeleaf will look these names that html in `webapp/views/` directory and append `.html` suffix for you.

Example

```
@RequestMapping(value = "/")
public String homePage() {
    return "foo/my-index";
}
```

Thymeleaf will be looking html named `my-index.html` in `webapp/views/foo/` directory. According to example configuration above.

Using checkboxes

Example method in controller

```
@RequestMapping(value = "/test")
public String showCheckbox(Model model) {
    boolean myBooleanVariable = false;
    model.addAttribute("myBooleanVariable", myBooleanVariable);
    return "sample-checkbox";
}
```

View: sample-checkbox.html

```
<input
      type="checkbox"
      name="myBooleanVariable"
      th:checked="${myBooleanVariable}"/>
```

Do not use `th:name` for checboxes, just `name`

Ajax form submition with Jquery

To submit form via Ajax with Jquery :

```
    <div id="yourPanel" th:fragment="yourFragment">
        <form id="yourForm" method="POST"
              th:action="@{/actions/postForm}"
              th:object="${yourFormBean}">
        <div class="form-group">
            <label for="param1"></label>
            <input class="form-component" type="text" th:field="*{param1}" />
        </div>
        <div class="form-group">
            <label for="param2"></label>
            <input class="form-component" type="text" th:field="*{param2}" />
        </div>
        <div class="form-group">
            <label for="param3"></label>
            <input class="form-component" type="checkbox" th:field="*{param3}" />
        </div>

        <button type="submit" class="btn btn-success">Save</button>
        <a href='#' class="btn btn-default">Cancel</a>
    </form>
    </div>

<script th:inline="javascript">
    /*<![CDATA[*/
    $(document).ready(function () {
        /*[+
         var postUrl = [[@{/actions/postForm(
         additionalParam=${#httpServletRequest.getParameter('additionalParam')}
         )}]];
         +]*/
        $("#yourForm").submit(function (e) {
            e.preventDefault();
            $.post(postUrl,
                    $(this).serialize(),
                    function (response) {
                        var isErr = 'hasError';
                        // when there are an error then show error
                        if (response.indexOf(isErr) > -1) {
                            $("#yourPanel").html(response);
                        } else {
                            var formData = $("#yourForm").serializeArray(),
                                    len = formData.length,
                                    urlEnd = '';
                            for (i = 0; i < len; i++) {
                                urlEnd += formData[i].name + '=' +
encodeURIComponent(formData[i].value) + '&';
                            }
                            /*[+
                             var urlReplacement = [[@{/another/page(

additionalParam=${#httpServletRequest.getParameter('additionalParam')}
                             )}]] + urlEnd;
                             +]*/

                            window.location.replace(urlReplacement);
                        }
                    }
            );
            return false;
        });
    });
    /*]]>*/
</script>
```

YourFormBean class :

```
@lombok.Getter
@lombok.Setter
@lombok.NoArgsConstructor
public class YourFormBean {
    private String param1;
    private String param2;
    private boolean param3;
}
```

Controller code :

```
@RequestMapping(value = "/actions/postForm", method = RequestMethod.POST)
public String saveForm(Model model,
        @RequestParam("additionalParam") Integer additionalParam,
        @Valid @ModelAttribute("yourFormBean") YourFormBean yourFormBean,
        BindingResult bindingResult,
        RedirectAttributes redirectAttributes) {
    if (bindingResult.hasErrors()) {
        model.addAttribute("hasError", true);
        return "your/template :: yourFragment";
    }
    redirectAttributes.addAttribute("additionalParam", additionalParam);

    return "redirect:/another/page";
}
```

Replacing fragments with ajax

If you want to replace parts of your website, ajax is an easy way to do it.

The **website.html** where you want to replace the content based on the selected value:

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

    <head>
```

```
        <title>Index</title>
    </head>

    <body>
        <select id="selection">
            <option>Content 1</option>
            <option>Content 2</option>
        </select>

        <div id="replace_div">
            Content goes here
        </div>

        <!-- JQury from Google CDN -->
        <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>

        <script>
            $(document).ready(function () {

                //call function when page is loaded
                getContent();

                //set on change listener
                $('#selection').change(getContent);

                function getContent() {

                    //create url to request fragment
                    var url = /content/;
                    if ($('#selection').val() === "Content 1") {
                        url = url + "content1";
                    } else {
                        url = url + "content2";
                    }

                    //load fragment and replace content
                    $('#replace_div').load(url);
                }
            })
        </script>
    </body>
</html>
```

And the **content.html** with the fragments you want to include based on the selected value:

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

    <body>
        <div th:fragment="content1">
            This is Content 1
        </div>

        <div th:fragment="content2">
            This is Content 2
         </div>
    </body>
</html>
```

Last but not least the Spring MVC **ContentController.java**:

```
@Controller
@RequestMapping("content")
public class ContentController {

    @RequestMapping("")
    public String loadContent() {
        return "website";
    }

    @RequestMapping("content1")
    public String getContent1() {
        return "content :: content1";
    }

    @RequestMapping("content2")
    public String getContent2() {
        return "content :: content2";
    }
}
```

Form Submission

Form object

```
package formSubmission;

public class Person {

    private String name;
    private int age;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name= name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }

}
```

Controller

```
package formSubmission;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

@Controller
public class FriendsController {

    @GetMapping("/friends")
    public String friendForm(Model model) {
        model.addAttribute("personForm", new Person());
        return "friendsForm";
    }

    @PostMapping("/friends")
    public String submissionResult(@ModelAttribute("personForm") Person person) {
        return "result";
    }

}
```

friendsForm.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Friend form</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
    <h1>Friend Form</h1>
    <form th:action="@{/friends}" th:object="${personForm}" method="post">
        <p>Name: <input type="text" th:field="*{name}"/></p>
        <p>Age: <input type="number" th:field="*{age}"/></p>
        <p><input type="submit" value="Submit"/></p>
    </form>
</body>
</html>
```

result.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Submission result</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
    <h1>th:text="'My friend ' + ${personForm.name} + ' is ' + ${personForm.age} + ' years 
old'"</h1>
</body>
</html>
```

Read Getting started with thymeleaf online: https://riptutorial.com/thymeleaf/topic/1895/getting-started-with-thymeleaf

Chapter 2: Expression Utility Objects

Examples

Format date

```
<p>
  Today: <span th:text="${#calendars.format(today,'dd MMMM yyyy')}">30 May 2017</span>
</p>
```

String length

```
<div th:if="*{userMessage!=null and #strings.length(userMessage)>0}">
     <label th:text = "*{userMessage}"/>
</div>
```

String contains

```
<div th:if="${#strings.contains(#httpServletRequest.requestURI, 'email')}">
    <div th:replace="fragments/email::welcome">
</div>
```

Parsing date

Get year from date

```
<p>
  Year: <span th:text="${#dates.year(today)}">2017</span>
</p>
```

Get month

```
<p>
  Month number: <span th:text="${#dates.month(today)}">8</span>
  Month: <span th:text="${#dates.monthName(today)}">August</span>
  Month short name: <span th:text="${#dates.monthNameShort(today)}">Aug</span>
</p>
```

Get day

```
<p>
  Day: <span th:text="${#dates.day(today)}">26</span>
</p>
```

Get day of week

```
<p>
```

```
    Day: <span th:text="${#dates.dayOfWeek(today)}">1</span>
    Day: <span th:text="${#dates.dayOfWeekName(today)}">Monday</span>
    Day: <span th:text="${#dates.dayOfWeekNameShort(today)}">Mo</span>
</p>
```

Get time

```
<p>
    Hour: <span th:text="${#dates.hour(today)}">10</span>
    Minute: <span th:text="${#dates.minute(today)}">50</span>
    Second: <span th:text="${#dates.second(today)}">48</span>
    Millisecond: <span th:text="${#dates.millisecond(today)}">48</span>
<p>
```

Format decimal

```
<p>
    Order sum: <span th:text="${#numbers.formatDecimal(orderSum, 0, 'COMMA', 2,
'POINT')}">1,145,000.52</span>
</p>
```

Read Expression Utility Objects online: https://riptutorial.com/thymeleaf/topic/10675/expression-utility-objects

Chapter 3: Externalizing Text in Thymeleaf

Examples

Localization

1. Create files for your messages

```
messages.properties
messages_en.properties
messages_fr.properties
...
```

2. Write messages in this files like this

```
header.label.title=Title
```

3. Configure path to this files (in this case in folder D:/project/messages) in application properties like:

```
messages.basename.path=D:/project/messages/messages
```

4. Configure MessageSource

```
@Value("${messages.basename.path}")
private String messagesBasename;

@Bean
public MessageSource messageSource() {
       ReloadableResourceBundleMessageSource messageSource = new
ReloadableResourceBundleMessageSource();
       messageSource.setFallbackToSystemLocale(false);
       messageSource.setBasenames("file:" + messagesBasename);
       return messageSource;
   }
```

5. Use messages on pages

```
<p th:text="#{header.label.title}">Title</p>
```

Localization messages with parameters

Write message in messages.properties

```
welcome.message=Hello, {0}!
```

Replace {0} with the user name inside thymeleaf tag

```
<h3 th:text="#{welcome.message(${some.variable})}">Hello, Placeholder</h3>
```

Read Externalizing Text in Thymeleaf online:
https://riptutorial.com/thymeleaf/topic/10668/externalizing-text-in-thymeleaf

Chapter 4: Spring Security and Thymeleaf

Examples

Secure your WebApp with Login and Logout

This example is a very simple Spring Boot application.

Maven Dependencies

At first add the following dependencies to your project. Spring Initializr is recommended when you create a new project.

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.1.RELEASE</version>
    <relativePath/>
</parent>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.thymeleaf.extras</groupId>
        <artifactId>thymeleaf-extras-springsecurity4</artifactId>
    </dependency>
</dependencies>
```

Create your WebApp

Create a web application with websites and controller. For example this very small webapp with just one page (index.html) and an entry for the login page.

```
@Configuration
public class MvcConfig extends WebMvcConfigurerAdapter{

    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addRedirectViewController("/", "index");
        registry.addViewController("/index").setViewName("index");
        registry.addViewController("/login").setViewName("login");
    }
}
```

Secure your WebApp

Configure Spring Security to secure your webapp. Eg. allow any request by authenticated users only. Allow static resources like js and css, otherwise they won't be loaded for non-authenticated users. Exclude the login & logout page from this rule and create a test user:

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
                .antMatchers("/css/*.css", "/js/*.js").permitAll()
                .anyRequest().authenticated()
                .and()
                .formLogin()
                .loginPage("/login")
                .permitAll()
                .and()
                .logout()
                .permitAll();
    }

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
                .withUser("user").password("password").roles("USER");
    }
}
```

Create the login page

The login page needs to have a form that makes a post request to "/login":

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

    <head>
        <title>Login</title>
        <link th:href="@{/css/stylesheet.css}" rel="stylesheet" type="text/css"/>
        <script type="text/javascript" th:src="@{/js/login.js}"></script>
    </head>
    <body>
        <!-- show notification on error -->
        <div th:if="${param.error}">
```

```
            Invalid username or password.
        </div>

        <!-- show notification of logout -->
        <div th:if="${param.logout}">
            You have been logged out.
        </div>

        <!-- login form -->
        <div>
            <form th:action="@{/login}" method="post">
                <h2 >Please sign in</h2>
                <label >User Name</label>
                <input type="text" name="username" th:required="required"
th:autofocus="autofocus"/>
                <br/>

                <label>Password</label>
                <input type="password" name="password" th:required="required" />
                <br/>

                <input type="submit" value="Sign In"/>
            </form>
        </div>
    </body>
</html>
```

When the user enters the wrong username/password the error parameter is set. When the user logs out the logout parameter is set. This is used to show the corresponding messages.

Access user properties

After a successful login the user is directed to the **index.html**. The Spring Security Dialect allows us to access the user properties like his username:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>Index</title>
    </head>
    <body>
        <div>
            <h3>Welcome <span th:text="${#authentication.name}"/></h3>
            <form th:action="@{/logout}" method="post">
                <input type="submit" value="Logout"/>
            </form>
        </div>
    </body>
</html>
```

A logout is achieved via post request to "/logout"

Displaying something for authenticated users only

```
<div sec:authorize="isAuthenticated()">
    This text is displayed for authenticated users.
</div>
```

Display username

You can show username for autenticated users

```
<div sec:authorize="isAuthenticated()">
    Welcome, <span sec:authentication="name">Username</span>
</div>
```

Display different content to different roles

The sec:authorize attribute renders its content when the attribute expression is evaluated to true

```
<div sec:authorize="hasRole('ROLE_ADMIN')">
    Content for administrators
</div>
<div sec:authorize="hasRole('ROLE_USER')">
    Content for users
</div>
```

The sec:authentication attribute is used to print logged user roles:

```
Roles: <span sec:authentication="principal.authorities">ROLE_USER, ROLE_ADMIN</span>
```

Read Spring Security and Thymeleaf online: https://riptutorial.com/thymeleaf/topic/9190/spring-security-and-thymeleaf

Chapter 5: Using Lists with Thymeleaf

Examples

Using list in select

You can use list variable to form `<select>` elements

```
 <select th:field="*{countries}">
     <option th:each="country: ${countries}"
             th:value="${country.id}"
             th:text="#{${'selected.label.' + country.name}}"/>
</select>
```

Form table

```
<table id="countryList">
    <thead>
        <tr>
            <th th:text="#{country.label.name}"> Country </th>
            <th th:text="#{country.label.capital}"> Capital </th>
            <th th:text="#{country.label.square}"> Square </th>
        </tr>
    </thead>
    <tbody>
        <tr th:each="country : ${countryList}">
            <td th:text="${country.name}"></td>
            <td th:text="${country.capital}"></td>
            <td th:text="${country.square}"></td>
        </tr>
    </tbody>
</table>
```

Read Using Lists with Thymeleaf online: https://riptutorial.com/thymeleaf/topic/10676/using-lists-with-thymeleaf

Credits

S. No	Chapters	Contributors
1	Getting started with thymeleaf	Aboodz, Alexander, Community, guille11, Jakub Pomykała, Maciek Łoziński, ppeterka, Prabhat, Rob Streeter, sanluck
2	Expression Utility Objects	Elizabeth
3	Externalizing Text in Thymeleaf	Elizabeth
4	Spring Security and Thymeleaf	Alexander, Elizabeth, Sébastien Temprado
5	Using Lists with Thymeleaf	Elizabeth