EBook Gratis

APRENDIZAJE thymeleaf

Free unaffiliated eBook created from
Stack Overflow contributors.

#thymeleaf

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: thymeleaf

It is an unofficial and free thymeleaf ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official thymeleaf.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con thymeleaf

Observaciones

Thymeleaf es un motor de plantillas, una biblioteca escrita en Java. Permite a un desarrollador definir una plantilla de página HTML, XHTML o HTML5 y luego rellenarla con datos para generar la página final. Por lo tanto, realiza una parte de *vista* de modelo de un patrón de vista- modelo.

El importante principio de diseño de Thymeleaf es que la propia plantilla debe estar correctamente escrita (X) HTML.

Versiones

Versión	Fecha	Último lanzamiento	Fecha
3.xx	2016-05-08	3.0.6	2017-05-07
2.xx	2012-02-09	2.1.5	2016-07-11

Examples

Configuración

Para comenzar con Thymeleaf visite la página de descarga oficial .

Dependencia maven

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

Dependencia de Gradle

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

Ejemplo de configuración

A partir de la versión 3.0, Thymeleaf solo admite la configuración de Java.

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

```
    return resolver;
}
```

En el método `viewResolver()` puede configurar, por ejemplo, la codificación y el tipo de contenido para las vistas. más información

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

En `templateEngine()` , puede agregar dialectos personalizados. Por ejemplo, para agregar el dialecto Spring Security, puede hacer esto como este `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;
}
```

Busque en el configurador el prefijo y el sufijo en el método `templateResolver()` . Le dice a Thymeleaf que, cada vez que el controlador devuelva vista, Thymeleaf buscará estos nombres que html en `webapp/views/` directory y `.html` sufijo `.html` .

Ejemplo

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

Thymeleaf buscará html llamado `my-index.html` en el directorio `webapp/views/foo/` . Según la configuración de ejemplo anterior.

Usando casillas de verificación

Ejemplo de método en el controlador

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

Ver: sample-checkbox.html

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

No use `th:name` para checboxes, solo `name`

Ajax forma la sumisión con Jquery.

Para enviar el formulario a través de Ajax con 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>
```

Clase de YourFormBean:

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

Código del controlador:

```
@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";
}
```

Sustitución de fragmentos por ajax.

Si desea reemplazar partes de su sitio web, ajax es una forma fácil de hacerlo.

El **sitio web.html** en el que desea reemplazar el contenido según el valor seleccionado:

```
<!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>
```

Y el **content.html** con los fragmentos que desea incluir en función del valor seleccionado:

```
<!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>
```

Por último, pero no menos importante, 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";
    }
}
```

Envío de formulario

Forma objeto

```
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;
    }

}
```

Controlador

```
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>
```

resultado.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>
```

Lea Empezando con thymeleaf en línea:
https://riptutorial.com/es/thymeleaf/topic/1895/empezando-con-thymeleaf

Capítulo 2: Externalizando Texto en Thymeleaf

Examples

Localización

1. Crea archivos para tus mensajes.

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

2. Escribe mensajes en estos archivos como este.

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

3. Configure la ruta de acceso a estos archivos (en este caso en la carpeta D: / project / messages) en las propiedades de la aplicación como:

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

4. Configurar 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. Usa mensajes en las páginas

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

Mensajes de localización con parámetros.

Escribir mensaje en messages.properties

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

Reemplace {0} con el nombre de usuario dentro de la etiqueta thymeleaf

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

Lea Externalizando Texto en Thymeleaf en línea:
https://riptutorial.com/es/thymeleaf/topic/10668/externalizando-texto-en-thymeleaf

Capítulo 3: Objetos de utilidad de expresión

Examples

Fecha de formato

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

Longitud de la cuerda

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

Cadena contiene

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

Fecha de análisis

Obtener el año a partir de la fecha

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

Obtener mes

```
<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>
```

Obtener dia

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

Obtener dia de la semana

```
<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>
```

Consigue tiempo

```
<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>
```

Formato decimal

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

Lea Objetos de utilidad de expresión en línea:
https://riptutorial.com/es/thymeleaf/topic/10675/objetos-de-utilidad-de-expresion

Capítulo 4: Seguridad de primavera y Thymeleaf

Examples

Asegure su aplicación web con inicio de sesión y cierre de sesión

Este ejemplo es una aplicación Spring Boot muy simple.

Dependencias Maven

Al principio agrega las siguientes dependencias a tu proyecto. Se recomienda Spring Initializr cuando cree un nuevo proyecto.

```
<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>
```

Crea tu aplicación web

Crear una aplicación web con sitios web y controlador. Por ejemplo, esta aplicación web muy pequeña con una sola página (index.html) y una entrada para la página de inicio de sesión.

```
@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");
    }
}
```

Asegure su aplicación web

Configure Spring Security para asegurar su aplicación web. P.ej. Permitir cualquier solicitud únicamente por usuarios autenticados. Permita recursos estáticos como js y css, de lo contrario no se cargarán para usuarios no autenticados. Excluya la página de inicio y cierre de sesión de esta regla y cree un usuario de prueba:

```
@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");
    }
}
```

Crear la página de inicio de sesión.

La página de inicio de sesión debe tener un formulario que realice una solicitud de publicación a "/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>
```

Cuando el usuario ingresa el nombre de usuario / contraseña incorrecto, se establece el parámetro de error. Cuando el usuario cierra la sesión se establece el parámetro de cierre de sesión. Esto se utiliza para mostrar los mensajes correspondientes.

Acceder a las propiedades del usuario

Después de un inicio de sesión exitoso, el usuario es dirigido al **index.html** . Spring Security Dialect nos permite acceder a las propiedades del usuario como su nombre de usuario:

```
<!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>
```

Un cierre de sesión se logra a través de la solicitud posterior a "/ logout"

Viendo algo solo para usuarios autenticados

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

Mostrar nombre de usuario

Puede mostrar el nombre de usuario para usuarios autenticados

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

Mostrar diferentes contenidos a diferentes roles

El atributo sec: authorize representa su contenido cuando la expresión del atributo se evalúa como verdadera

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

El atributo sec: authentication se utiliza para imprimir roles de usuario registrados:

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

Lea Seguridad de primavera y Thymeleaf en línea:
https://riptutorial.com/es/thymeleaf/topic/9190/seguridad-de-primavera-y-thymeleaf

Capítulo 5: Usando Listas con Thymeleaf

Examples

Usando la lista en seleccionar

Puede utilizar la variable list para formar elementos `<select>`

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

Tabla de formularios

```
<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>
```

Lea Usando Listas con Thymeleaf en línea:
https://riptutorial.com/es/thymeleaf/topic/10676/usando-listas-con-thymeleaf

Creditos

S. No	Capítulos	Contributors
1	Empezando con thymeleaf	Aboodz, Alexander, Community, guille11, Jakub Pomykała, Maciek Łoziński, ppeterka, Prabhat, Rob Streeter, sanluck
2	Externalizando Texto en Thymeleaf	Elizabeth
3	Objetos de utilidad de expresión	Elizabeth
4	Seguridad de primavera y Thymeleaf	Alexander, Elizabeth, Sébastien Temprado
5	Usando Listas con Thymeleaf	Elizabeth