

INGENIERÍA WEB Y COMPUTACIÓN EN LA NUBE

Bloque3: Parte servidora (backend)

TEMA 3.2: APLICACIONES WEB CON SPRING
MVC Y THYMELEAF

Boni García

boni.garcia@urjc.es



Índice de contenidos

1. Spring MVC
2. Thymeleaf
3. Soporte de internacionalización (I18N)
4. Envío de información al servidor
5. Gestión de datos de sesión

Índice de contenidos

1. Spring MVC

- Introducción
- Controladores
- Vistas
- Ejemplo

2. Thymeleaf

3. Soporte de internacionalización (I18N)

4. Envío de información al servidor

5. Gestión de datos de sesión

1. Spring MVC

Introducción

- El modelo vista controlador (MVC) que separa que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario
- El esquema del patrón MVC en Spring es:



<http://spring.io/guides/gs/serving-web-content/>

1. Spring MVC

Controladores

- Los controladores (*Controllers*) son clases Java encargadas de atender las peticiones web
- Procesan los datos que llegan en la petición (parámetros)
- Hacen peticiones a la base de datos, usan diversos servicios, etc...
- Definen la información que será visualizada en la página web (el modelo)
- Determinan que vista será la encargada de generar la página HTML

1. Spring MVC

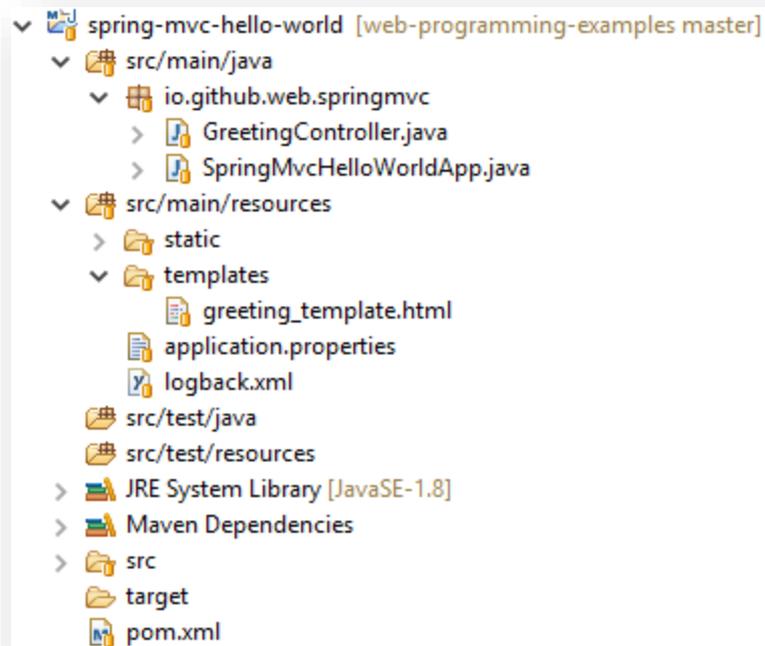
Vistas

- Las vistas en Spring MVC se implementan como tecnologías de plantillas HTML
- Generan HTML partiendo de una plantilla y la información que viene del controlador (modelo)
- Existen diversas tecnologías de plantillas que se pueden usar con Spring MVC: JSP, Thymeleaf, FreeMarker, etc...
- Nosotros usaremos **Thymeleaf**

1. Spring MVC

Ejemplo

- Estructura de la aplicación vista desde Eclipse:



1. Spring MVC

Ejemplo

Proyecto padre del que se hereda la configuración (Spring Boot)

Java 8

Tipo de proyecto Spring Boot

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>io.github.web</groupId>
  <artifactId>spring-mvc-hello-world</artifactId>
  <version>1.0.0</version>

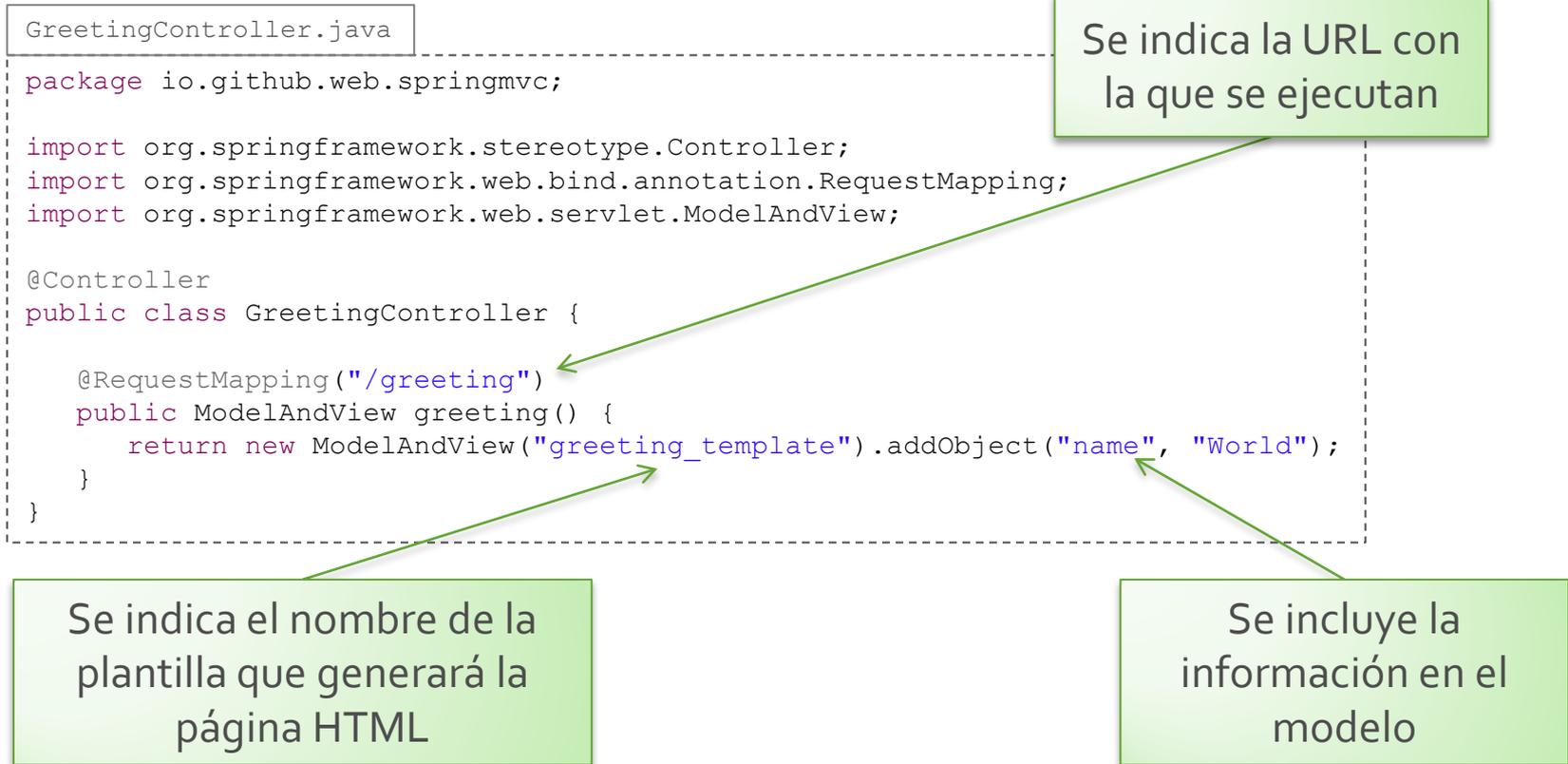
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.2.7.RELEASE</version>
  </parent>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <java.version>1.8</java.version>
  </properties>

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

1. Spring MVC

Ejemplo



1. Spring MVC

Ejemplo

- Vistas (implementado con Thymeleaf)

```
greeting_template.html
```

```
<html xmlns:th="http://www.thymeleaf.org">
<body>
  <p>Hello <span th:text="${name}">Friend</span>!</p>
</body>
</html>
```

Se accede a los objetos que el **controlador** ha puesto en el **modelo**

El valor del objeto *name* sustituirá a la palabra *Friend* cuando se genere el HTML

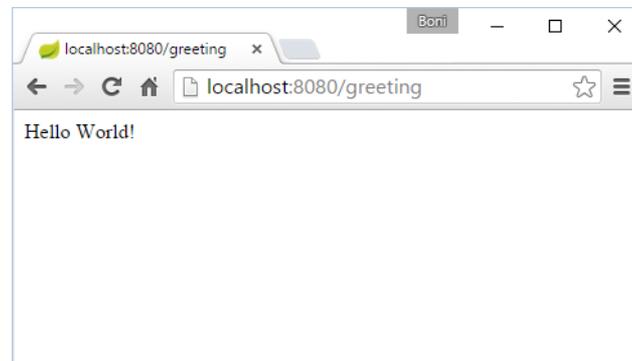
APLICACIONES WEB CON SPRING MVC Y THYMELEAF

1. Spring MVC

Ejemplo

- La aplicación se ejecuta como una app Java normal
- En Eclipse: botón derecho proyecto > Run as... > Java Application...

```
SpringMvcHelloWorldApp.java  
  
@SpringBootApplication  
public class SpringMvcHelloWorldApp extends WebMvcConfigurerAdapter {  
  
    public static void main(String[] args) {  
        SpringApplication.run(SpringMvcHelloWorldApp.class, args);  
    }  
}
```



Índice de contenidos

1. Spring MVC
2. Thymeleaf
 - Introducción
 - Ejemplo
 - Manejando texto
 - Condicionales
 - Iteraciones
3. Envío de información al servidor
4. Gestión de datos de sesión

2. Thymeleaf

Introducción

- Spring MVC se apoya en alguna tecnología de plantillas para la generación de páginas HTML:

<FreeMarker>

<http://freemarker.org/>



<http://www.oracle.com/technetwork/java/javasee/jsp/>



<http://www.thymeleaf.org/>



<http://velocity.apache.org/>

2. Thymeleaf

Introducción

- Todas estas tecnologías son conceptualmente similares
 - Los ficheros HTML se generan con **plantillas** que contienen código HTML junto con referencias a variables y funciones
 - Ejemplo implementado con FreeMarker:

```

<html>
<body>
  <p>Hello ${name}!</p>
</body>
</html>
    
```

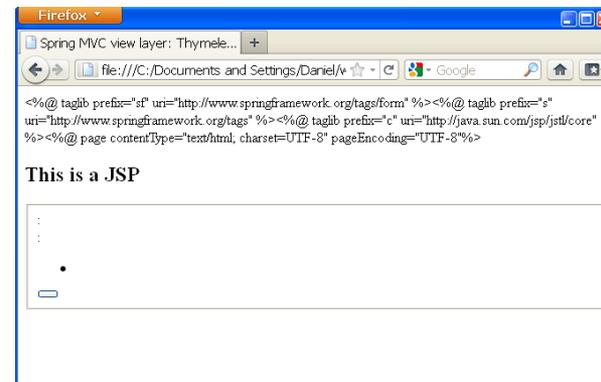
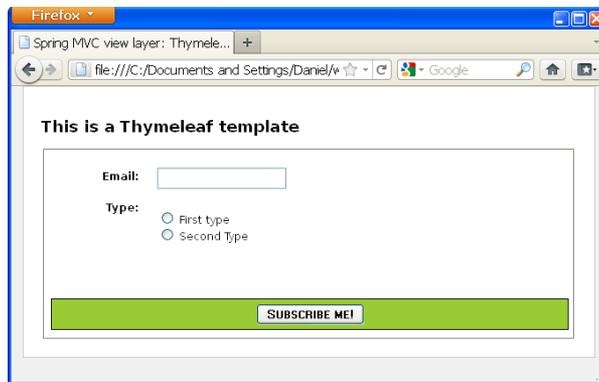
- **Thymeleaf** se diferencia de las demás en que las plantillas son ficheros HTML válidos que pueden verse en un navegador sin necesidad de servidor web (*natural templating*)
- Esta característica es ideal para la separación de roles: diseñadores y desarrolladores

APLICACIONES WEB CON SPRING MVC Y THYMELEAF

2. Thymeleaf

Introducción

- La tecnología de plantillas JSP (Java Server Page) es la más extendida
- Ambas pueden ser usadas en conjunción con Spring MVC
- Thymeleaf permite realizar la maquetación HTML sin necesidad de que intervenga el servidor



- Más información: <http://www.thymeleaf.org/doc/articles/thvsjsp.html>

2. Thymeleaf

Introducción

- Thymeleaf está totalmente Integrado con Spring (MVC, Security)
- Soporta de dos tipo de lenguajes de expresiones (EL, *Expression Language*) para el acceso a objetos Java:
 - OGNL (*Object-Graph Navigation Language*):

```
#{myObject.property}
```

```
#{myObject.method() }
```

- Spring EL (*Spring Expression Language*)

```
#{@myBean.method() }
```

- Tutorial: <http://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html>

2. Thymeleaf

Introducción

- La sintaxis de las plantillas Thymeleaf se define en las páginas HTML mediante la etiqueta `th`
- Los navegadores ignorarán el espacio de nombre que no entienden (`th`) con lo que la página seguirá siendo válida

```

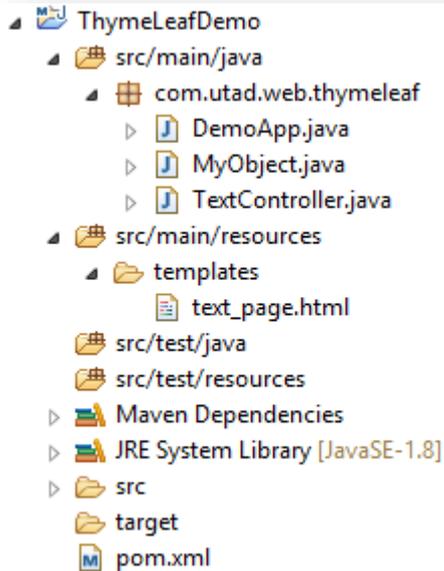
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<body>
  <p>Hello <span th:text="{name}"></span></p>
</body>
</html>
    
```

El atributo `xmlns` define el espacio de nombres (*XML Namespace*) para `th`.
 Un espacio de nombres permite definir nombres de elementos y atributos únicos en un documento XML (o HTML)

APLICACIONES WEB CON SPRING MVC Y THYMELEAF

2. Thymeleaf

Ejemplo



pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>io.github.bonigarcia.web</groupId>
  <artifactId>ThymeLeafDemo</artifactId>
  <version>1.0.0</version>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.2.7.RELEASE</version>
  </parent>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <java.version>1.8</java.version>
  </properties>

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

Fork me on GitHub

APLICACIONES WEB CON SPRING MVC Y THYMELEAF

2. Thymeleaf

Ejemplo

MyObject.java

```
package io.github.web.thymeleaf;

public class MyObject {

    private String name;
    private String description;

    public MyObject(String name, String description) {
        this.name = name;
        this.description = description;
    }

    public String sayHello() {
        return "Hello!!!";
    }
    // Getters and setters
}
```

SpringMvcThymeleafApp.java

```
package io.github.web.thymeleaf;

import org.springframework.boot.*;
import org.springframework.web.servlet.config.annotation.*;

@SpringBootApplication
public class SpringMvcThymeleafApp extends WebMvcConfigurerAdapter {

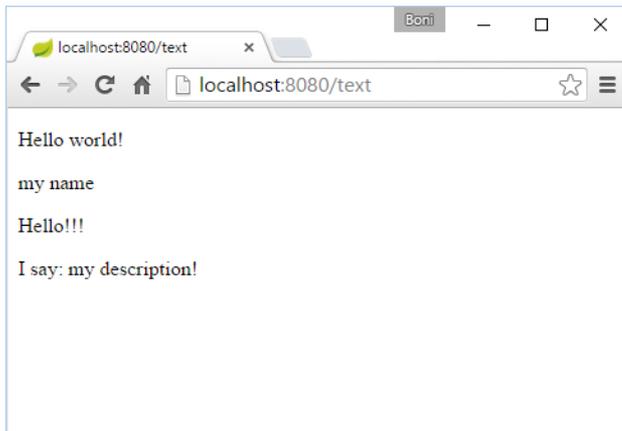
    public static void main(String[] args) {
        SpringApplication.run(SpringMvcThymeleafApp.class, args);
    }
}
```

APLICACIONES WEB CON SPRING MVC Y THYMELEAF

2. Thymeleaf

Manejando texto

- Para mostrar texto en la plantilla usamos la etiqueta `th:text`



TextController.java

```
package io.github.web.thymeleaf;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.servlet.ModelAndView;  
  
@Controller  
public class TextController {  
    @RequestMapping("/text")  
    public ModelAndView text() {  
        MyObject myObject = new MyObject("my name", "my description");  
        return new ModelAndView("text_page").addObject("greetings",  
            "Hello world!").addObject("myobj", myObject);  
    }  
}
```

text_page.html

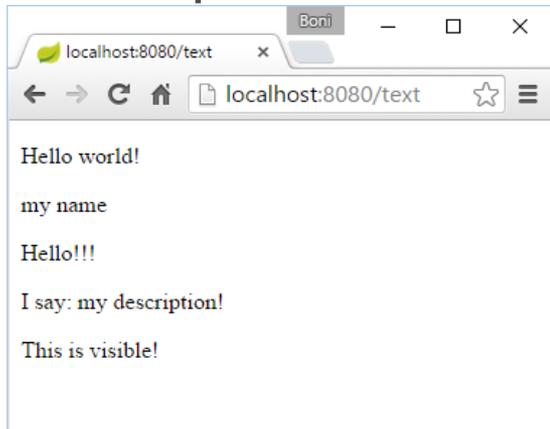
```
<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
  <body>  
    <p th:text="${greetings}"></p>  
    <p th:text="${myobj.name}"></p>  
    <p th:text="${myobj.sayHello()}"></p>  
    <p th:text="| I say: ${myobj.description}!"></p>  
  </body>  
</html>
```

APLICACIONES WEB CON SPRING MVC Y THYMELEAF

2. Thymeleaf

Condicionales

- Etiqueta `th:if`



TextController.java

```
package io.github.web.thymeleaf;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class TextController {

    @RequestMapping("/text")
    public ModelAndView text() {
        MyObject myObject = new MyObject("my name", "my description");
        return new ModelAndView("text_page")
            .addObject("greetings", "Hello world!")
            .addObject("myobj", myObject).addObject("hidden", false);
    }
}
```

text_page.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<body>
<p th:text="${greetings}"></p>
<p th:text="${myobj.name}"></p>
<p th:text="${myobj.sayHello()}"></p>
<p th:text="|I say: ${myobj.description}!!"></p>
<p th:if="${not hidden}">This is visible!</p>
</body>
</html>
```

APLICACIONES WEB CON SPRING MVC Y THYMELEAF

2. Thymeleaf

Iteraciones

- Etiqueta `th:each`

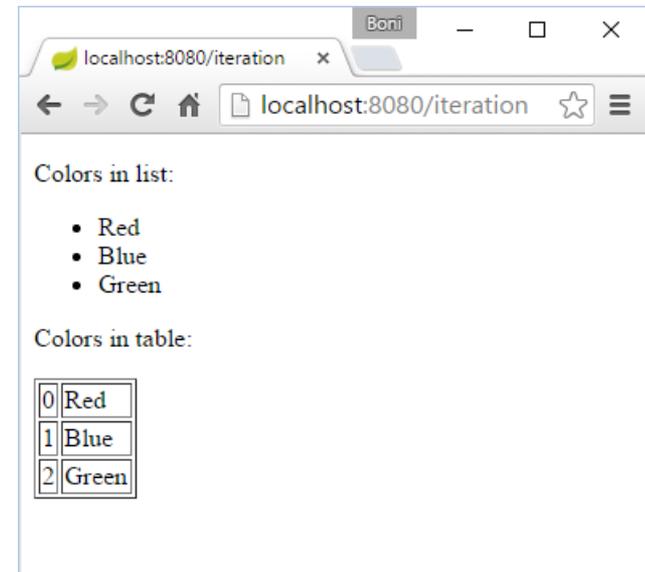
TextController.java

```
@RequestMapping("/iteration")
public ModelAndView iteration() {
    List<String> colors = Arrays.asList("Red", "Blue", "Green");
    return new ModelAndView("iteration_template").addObject("colors", colors);
}
```

iteration_template.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<body>
<p>Colors in list:</p>
<ul>
<li th:each="color : ${colors}" th:text="${color}">Color</li>
</ul>
<p>Colors in table:</p>
<table border="1">
<tr th:each="color, it : ${colors}">
<td th:text="${it.index}">1</td>
<td th:text="${color}">Color</td>
</tr>
</table>
</body>
</html>
```

Se puede declarar una variable adicional para guardar información de la iteración

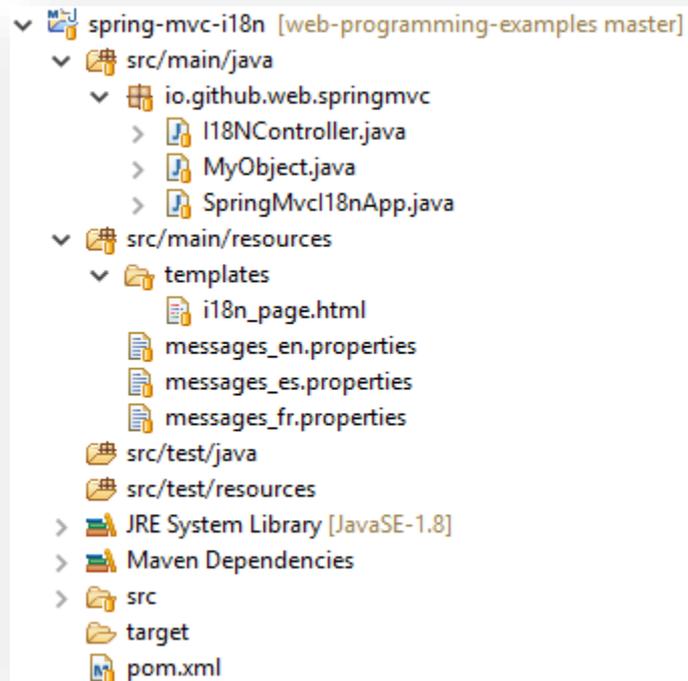


Índice de contenidos

1. Spring MVC
2. Thymeleaf
3. Soporte de internacionalización (I18N)
4. Envío de información al servidor
5. Gestión de datos de sesión

3. Soporte de internacionalización (I18N)

Fork me on GitHub



messages_en.properties

welcome=Welcome to my web!

messages_es.properties

welcome=Bienvenido a mi web!

messages_fr.properties

welcome=Bienvenue sur mon site web!

3. Soporte de internacionalización (I18N)

Nombre del fichero que contiene los mensajes de I18N

Región (locale) por defecto

Parámetro con el que cambiar la región desde URL

SpringMvcI18nApp.java

```
@SpringBootApplication
public class SpringMvcI18nApp extends WebMvcConfigurerAdapter {
    @Bean
    public MessageSource messageSource() {
        ResourceBundleMessageSource messageSource = new ResourceBundleMessageSource();
        messageSource.setBasename("messages");
        return messageSource;
    }
    @Bean
    public LocaleResolver localeResolver() {
        SessionLocaleResolver sessionLocaleResolver = new SessionLocaleResolver();
        sessionLocaleResolver.setDefaultLocale(Locale.ENGLISH);
        return sessionLocaleResolver;
    }
    @Bean
    public LocaleChangeInterceptor localeChangeInterceptor() {
        LocaleChangeInterceptor result = new LocaleChangeInterceptor();
        result.setParamName("lang");
        return result;
    }
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(localeChangeInterceptor());
    }
}
```

3. Soporte de internacionalización (I18N)

I18NController.java

```
package io.github.web.springmvc;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class I18NController {

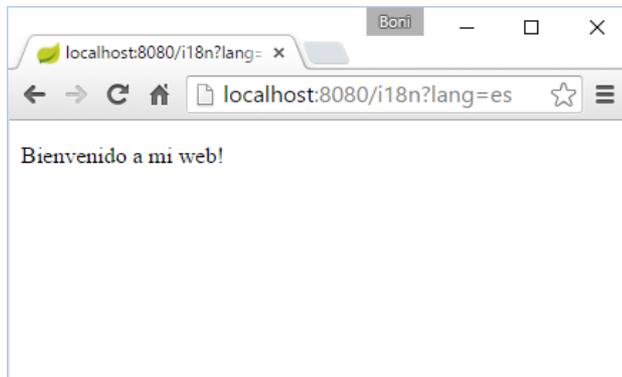
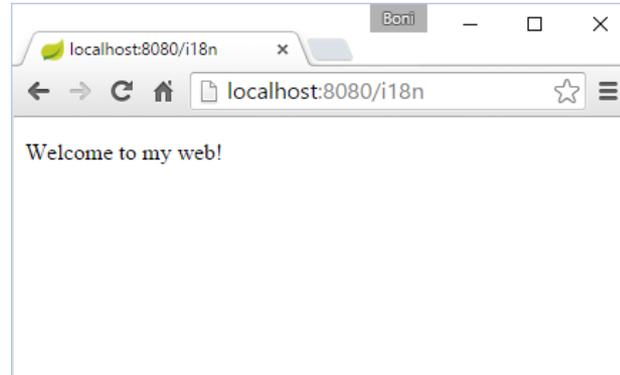
    @RequestMapping("/i18n")
    public ModelAndView i18n() {
        return new ModelAndView("i18n_page");
    }
}
```

i18n_page.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<body>
<p th:text="#{welcome}">Default greetings</p>
</body>
</html>
```

El símbolo # hace que la plantilla se rellene con los datos de I18N

3. Soporte de internacionalización (I18N)



Índice de contenidos

1. Spring MVC
2. Thymeleaf
3. Soporte de internacionalización (I18N)
4. Envío de información al servidor
 - Envío mediante formulario
 - Envío mediante URL
 - URLs en Thymeleaf
5. Gestión de datos de sesión

4. Envío de información al servidor

- Formas comunes de enviar información del navegador al servidor:
 - **Mediante formularios HTML:** La información la introduce manualmente el usuario
 - **Insertando información en la URL en enlaces:** La información la incluye el desarrollador para que esté disponible cuando el usuario pulsa el enlace
- Acceso a la información en el servidor
 - La información se envía como pares clave=valor
 - Se accede a la información como parámetros en los métodos del controlador

4. Envío de información al servidor

Envío mediante formulario

- Ejemplo de formulario HTML (parte cliente) y controlador (parte servidor):

```
<!DOCTYPE html>
<html>
<body>
  <form action="processForm" method="post">
    <h1>Form</h1>
    <label for="input">Input</label>
    <input type="text" name="input"
      id="input">
    <input type="submit">
  </form>
</body>
</html>
```

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class FormController {

    @RequestMapping("/processForm")
    public ModelAndView process(@RequestParam String input) {
        return new ModelAndView("result").addObject("result",
            input);
    }
}
```

- Recordatorio: Existen dos métodos para enviar datos desde un formulario HTML: GET y POST

4. Envío de información al servidor

Envío mediante URL (opción 1)

- La primera opción de enviar datos mediante URL consiste en simular el envío GET de un formulario:
 - Se incluyen al final de la URL separados con ? (*query*)
 - Los parámetros se separan entre sí con &
 - Cada parámetro se codifica como `nombre=valor`
- Ejemplo:

<http://my-server.com/path?option=web&view=category&lang=es>

4. Envío de información al servidor

Envío mediante URL (opción 1)

- Formato: Codificación de los nombres y los valores
 - Los caracteres alfanuméricos "a" hasta "z", "A" hasta "Z" y "0" hasta "9" se quedan igual
 - Los caracteres especiales ".", "-", "*", y "_" se quedan igual
 - El carácter espacio " " es convertido al signo "+" o con su valor hexadecimal %20
 - Todos los otros caracteres son codificados en uno o más bytes. Después cada byte es representado por la cadena de 3 caracteres "%xy", donde xy es la representación en hexadecimal
- Ejemplo:

<http://localhost:8080/demo?parameter=hello%20world>

4. Envío de información al servidor

Envío mediante URL (opción 1)

- Para acceder a la información se usa el mismo mecanismo que para leer los campos del formulario
- Ejemplo:

<http://my-server.com/path?option=web&view=category&lang=es>

```
@RequestMapping("/path")
public ModelAndView path(@RequestParam String option,
    @RequestParam String view, @RequestParam String lang) {
    // Create and return model
}
```

4. Envío de información al servidor

Envío mediante URL (opción 2)

- La información también se pueden incluir como parte de la propia URL, en vez de cómo parámetros
- Ejemplo:

<http://my-server.com/path/web/category/es>

```
@RequestMapping("/path/{option}/{view}/{lang}")  
public ModelAndView path(@PathVariable String option,  
    @PathVariable String view, @PathVariable String lang) {  
    // Create and return model  
}
```

4. Envío de información al servidor

URLs en Thymeleaf

- Debido a su importancia, las URLs son “ciudadanos de primera clase” en Thymeleaf y tienen una sintaxis especial
- Los enlaces también se pueden construir en una plantilla con la información del modelo
- Ejemplo: Modelo con un objeto *id* :

```

<!-- Will produce '/order/details?orderId=3' -->
<a href="details.html" th:href="@{/order/details(orderId=${id})}">view</a>

<!-- Will produce '/order/3/details' -->
<a href="details.html" th:href="@{/order/{orderId}/details(orderId=${id})}">view</a>
    
```

La etiqueta `th:href` generará el atributo `href` del enlace

Se usa con el símbolo `@`

Índice de contenidos

1. Spring MVC
2. Thymeleaf
3. Soporte de internacionalización (I18N)
4. Envío de información al servidor
5. Gestión de datos de sesión
 - Objeto `HttpSession`
 - Gestión de la sesión en Spring
 - Objeto `HttpSession` vs sesión Spring

5. Gestión de datos de sesión

- Es habitual que las aplicaciones web gestionen información diferente para cada usuario que está navegando. Por ejemplo:
 - Amigos en Facebook
 - Lista de correos en Gmail
 - Carrito de la compra en Amazon
- Se puede gestionar la información del usuario en dos ámbitos diferentes:
 - Información que se utiliza durante la navegación del usuario, durante la **sesión** actual
 - Información que se guarda mientras que el usuario no está navegando y que se recupera cuando el usuario vuelve a visitar la página web (**información persistente**)

5. Gestión de datos de sesión

- **Sesión:** Mantener información mientras el usuario navega por la web
 - Cuando el usuario pasa cierto tiempo sin realizar peticiones a la web, la sesión finaliza automáticamente (caducidad)
 - El tiempo para que caducidad es configurable
 - La información de sesión se guarda en memoria del servidor web
- **Información persistente:** Guardar información entre distintas navegaciones por la web
 - Para que podamos guardar información del usuario en el servidor, es necesario que el usuario se identifique al acceder a la página
 - La información se suele guardar en el servidor web en una BBDD
 - La lógica de la aplicación determina a qué información de la BBDD puede acceder cada usuario

5. Gestión de datos de sesión

- Existen dos técnicas principales:

1. Objeto `HttpSession`

- Es la forma básica de gestión de sesiones en Java EE
- Existe un objeto `HttpSession` por cada usuario que navega por la web
- Se puede almacenar información en una petición y recuperar la información en otra petición posterior
- Es de más bajo nivel

2. Componente específico para cada usuario

- Cada usuario guarda su información en uno o varios componentes Spring
- Existe una instancia por cada usuario (cuando lo habitual es tener una única instancia por componente)
- Es de más alto nivel

5. Gestión de datos de sesión

Objeto HttpSession

- La sesión se representa como un objeto del interfaz `javax.servlet.http.HttpSession`
- El framework Spring es el encargado de crear un objeto de la sesión diferente para cada usuario
- Para acceder al objeto de la sesión del usuario que está haciendo una petición, basta incluirlo como parámetro en el método del controlador

```

@RequestMapping("/mypath")
public ModelAndView process(HttpSession session, ...) {
    Object info = ...;
    session.setAttribute("info", info);
    return new ModelAndView("template");
}
    
```

5. Gestión de datos de sesión

Objeto HttpSession

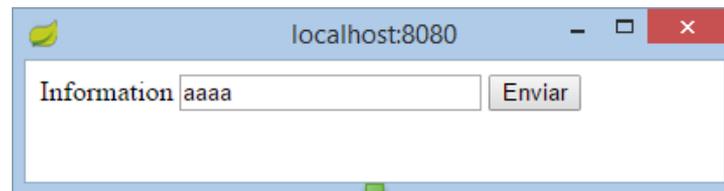
- Ejemplo:
 - La aplicación recoge la información de formulario y la guarda de dos formas:
 - Atributo del controlador (compartida)
 - Atributo de la sesión (usuario)
 - Una vez guardada la información, se puede acceder a ella y generar una página
 - Si dos usuarios visitan esta página a la misma vez, se puede ver cómo la información del controlador es compartida (la que guarda el último usuario es la que se muestra), pero la que se guarda en la sesión es diferente para cada usuario

APLICACIONES WEB CON SPRING MVC Y THYMELEAF

5. Gestión de datos de sesión

Objeto HttpSession

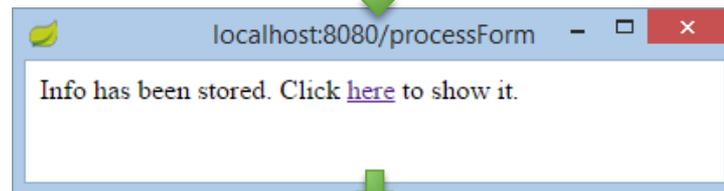
- Ejemplo:



localhost:8080

Information

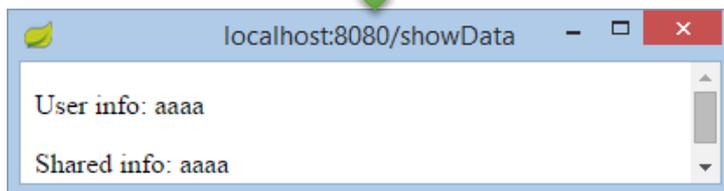
Guarda la información en la un **atributo de la sesión** y en un **atributo del controlador**



localhost:8080/processForm

Info has been stored. Click [here](#) to show it.

Recupera la información de la **sesión (usuario)** y del **controlador (compartida)**



localhost:8080/showData

User info: aaaa
Shared info: aaaa

5. Gestión de datos de sesión

Objeto HttpSession

- Ejemplo:

```
@Controller
public class SessionController {

    private String sharedInfo;

    @RequestMapping(value = "/processForm")
    public ModelAndView processForm(@RequestParam String info,
        HttpSession session) {
        session.setAttribute("userInfo", info);
        sharedInfo = info;
        return new ModelAndView("form_result");
    }

    @RequestMapping("/showData")
    public ModelAndView showData(HttpSession session) {
        String userInfo = (String) session.getAttribute("userInfo");
        return new ModelAndView("data").addObject("userInfo",
            userInfo).addObject("sharedInfo", sharedInfo);
    }
}
```

5. Gestión de datos de sesión

Objeto HttpSession

- Métodos de HttpSession:
 - `void setAttribute(String name, Object value)`: Asocia un objeto a la sesión identificado por un nombre
 - `Object getAttribute(String name)`: Recupera un objeto previamente asociado a la sesión
 - `boolean isNew()`: Indica si es la primera página que solicita el usuario (sesión nueva)
 - `void invalidate()`: Cierra la sesión del usuario borrando todos sus datos. Si visita nuevamente la página, será considerado como un usuario nuevo
 - `void setMaxInactiveInterval(int seconds)`: Configura el tiempo de inactividad para cerrar automáticamente la sesión del usuario.

5. Gestión de datos de sesión

Gestión de la sesión en Spring

- En Spring existe una forma de más **alto nivel** para asociar información al usuario
- Consiste en crear un `@Component` especial que se asociará a cada usuario y hacer `@Autowired` del mismo en el controlador que se utilice
- Internamente Spring hace bastante magia para que la información se gestione de forma adecuada

5. Gestión de datos de sesión

Gestión de la sesión en Spring

- Ejemplo:

```
import org.springframework.context.annotation.Scope;
import org.springframework.context.annotation.ScopedProxyMode;
import org.springframework.stereotype.Component;
import org.springframework.web.context.WebApplicationContext;

@Component
@Scope(value = WebApplicationContext.SCOPE_SESSION)
public class User {

    private String info;

    public void setInfo(String info) {
        this.info = info;
    }

    public String getInfo() {
        return info;
    }
}
```

La anotación
@Scope con estos
valores hace que
haya un componente
por cada usuario

5. Gestión de datos de sesión

Gestión de la sesión en Spring

- Ejemplo:

```

@Controller
public class SessionController2 {

    @Autowired
    private User user;

    private String sharedInfo;

    @RequestMapping(value = "/processForm")
    public ModelAndView processForm(@RequestParam String info) {
        user.setInfo(info);
        sharedInfo = info;
        return new ModelAndView("info_result");
    }

    @RequestMapping("/showData")
    public ModelAndView showData(HttpSession session) {
        String userInfo = user.getInfo();
        return new ModelAndView("data_page").addObject("userInfo",
            userInfo).addObject("sharedInfo", sharedInfo);
    }
}
    
```

5. Gestión de datos de sesión

Objeto `HttpSession` vs sesión Spring

- Ambas técnicas se pueden combinar
- El objeto `HttpSession` se utilizará para controlar el ciclo de vida de la sesión (si es nueva, invalidarla, etc...)
- El componente se usará para gestionar la información asociada al usuario