

Spring Framework

Spring MVC. Los controladores



EE. SS. M^a AUXILIADORA

2º DAM

Autor: Manuel Torres Molina

ACCESO A DATOS

Spring Framework

Spring MVC. Los controladores

Las vistas

Las distintas vistas se añaden en la ruta **src/main/resources**, donde en la carpeta **static** van los estilos e imágenes, y en la carpeta **templates** las distintas plantillas html.

Motor de plantillas Thymeleaf

Permite al servidor Spring boot montar las plantillas Web.

Se pueden añadir o modificar algunas de las propiedades del mismo añadiéndolas en el fichero **application.yml**. Estas configuraciones de Thymeleaf serían las siguientes:

```
spring.thymeleaf.cache=true # Enable template caching.
spring.thymeleaf.check-template=true # Check that the template exists
before rendering it.
spring.thymeleaf.check-template-location=true # Check that the
templates location exists.
spring.thymeleaf.content-type=text/html # Content-Type value.
spring.thymeleaf.enabled=true # Enable MVC Thymeleaf view resolution.
spring.thymeleaf.encoding=UTF-8 # Template encoding.
spring.thymeleaf.excluded-view-names= # Comma-separated list of view
names that should be excluded from resolution.
spring.thymeleaf.mode=HTML5 # Template mode to be applied to templates.
See also StandardTemplateModeHandlers.
spring.thymeleaf.prefix=classpath:/templates/ # Prefix that gets
prepending to view names when building a URL.
spring.thymeleaf.suffix=.html # Suffix that gets appended to view names
when building a URL.
spring.thymeleaf.template-resolver-order= # Order of the template
resolver in the chain.
spring.thymeleaf.view-names= # Comma-separated list of view names that
can be resolved.
```

Formas de retornar una plantilla

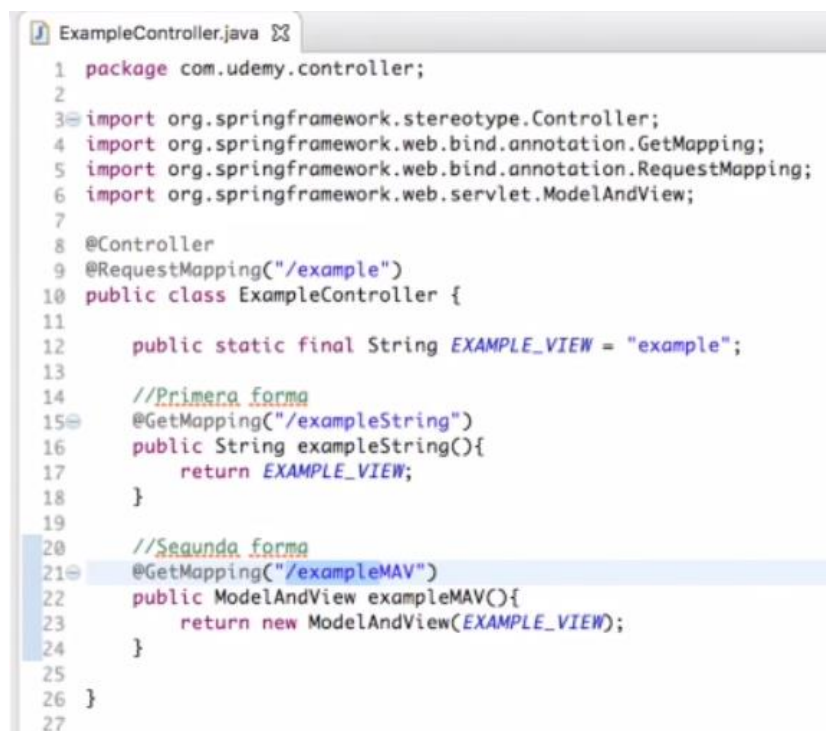
Creemos una vista de ejemplo llamada example.html dentro de la carpeta templates y una clase java dentro de un paquete controller creado dentro de src/main/java que nos va a servir de controlador llamado ExampleController.java.

Vista (example.html)

A screenshot of a code editor showing the content of a file named 'example.html'. The code is HTML and includes a DOCTYPE declaration, head section with charset and title, and a body section with a single line of text 'Hello World!'.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8"/>
5 <title>HELLO</title>
6 </head>
7 <body>
8 <h1>Hello World!</h1>
9 </body>
10 </html>
```

Controlador (ExampleController.java)

A screenshot of a code editor showing the content of a file named 'ExampleController.java'. The code is a Java class with package, imports, annotations, and two methods. The first method returns a static string, and the second method returns a ModelAndView object.

```
1 package com.udemy.controller;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RequestMapping;
6 import org.springframework.web.servlet.ModelAndView;
7
8 @Controller
9 @RequestMapping("/example")
10 public class ExampleController {
11
12     public static final String EXAMPLE_VIEW = "example";
13
14     //Primera forma
15     @GetMapping("/exampleString")
16     public String exampleString(){
17         return EXAMPLE_VIEW;
18     }
19
20     //Segunda forma
21     @GetMapping("/exampleMAV")
22     public ModelAndView exampleMAV(){
23         return new ModelAndView(EXAMPLE_VIEW);
24     }
25 }
26
27
```

Esta clase contiene las siguientes anotaciones:

- **@Controller** antes de la clase para indicar que es un controlador

- **@RequestMapping("/example")** antes de la clase indicando la ruta /example en su argumento para identificar a esa clase.
- **@GetMapping("/exampleString")** y **@GetMapping("/exampleMAV")** antes de los métodos para referenciarlos. La primera forma se suele utilizar cuando hay que insertar pocos datos en las plantillas o solo redireccionar. La segunda forma se utiliza cuando hay que meter muchos datos.

Al final indicaremos las siguientes rutas en el navegador para llegar a los métodos que llama a la plantilla de example.html.

<localhost:8080/example/exampleString>

<localhost:8080/example/exampleMAV>

Insertando datos simples en plantillas

Añadimos a la plantilla example.html el siguiente código con un atributo de thymeleaf:

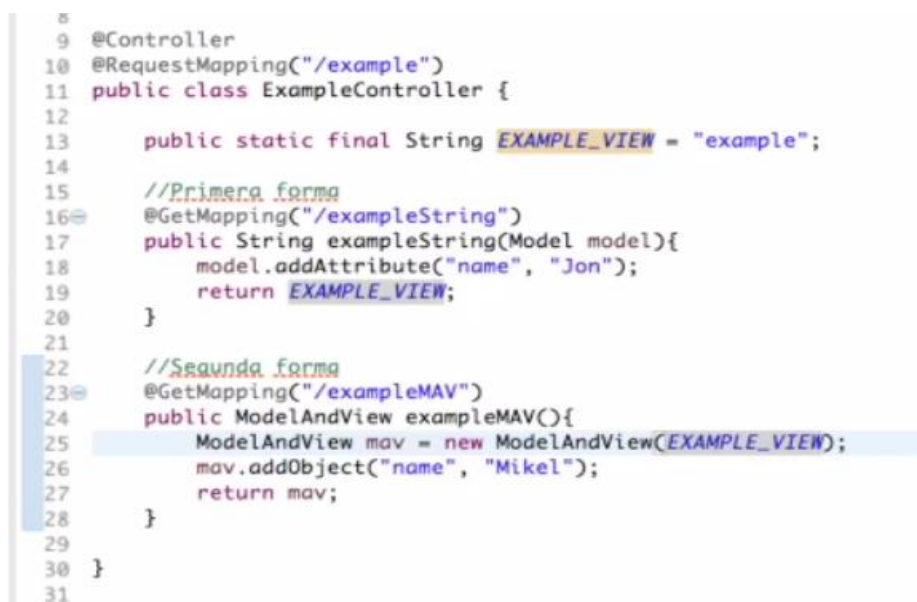


```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8"/>
5 <title>HELLO</title>
6 </head>
7 <body>
8 <h1>Hello <span th:text='${name}'></span> !!!!!</h1>
9 </body>
10 </html>

```

En el controlador tendremos que insertar este dato identificado por name en la plantilla:



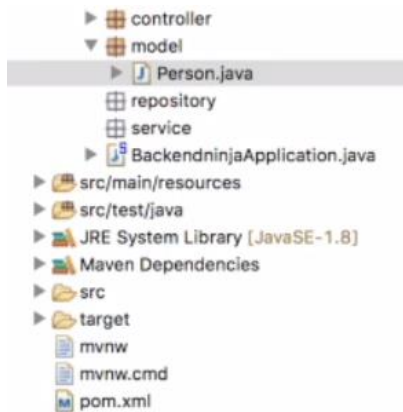
```

8
9 @Controller
10 @RequestMapping("/example")
11 public class ExampleController {
12
13     public static final String EXAMPLE_VIEW = "example";
14
15     //Primera forma
16     @GetMapping("/exampleString")
17     public String exampleString(Model model){
18         model.addAttribute("name", "Jon");
19         return EXAMPLE_VIEW;
20     }
21
22     //Segunda forma
23     @GetMapping("/exampleMAV")
24     public ModelAndView exampleMAV(){
25         ModelAndView mav = new ModelAndView(EXAMPLE_VIEW);
26         mav.addObject("name", "Mikel");
27         return mav;
28     }
29 }
30
31

```

Insertando datos complejos en una plantilla

Creamos un paquete **model**, dentro de **src/main/java** para crear una clase para nuestros datos complejos.



Esta clase se llama **Person.java** y tiene el siguiente código:

```
3 public class Person {
4
5     private String name;
6     private int age;
7
8     public String getName() {
9         return name;
10    }
11
12    public void setName(String name) {
13        this.name = name;
14    }
15
16    public int getAge() {
17        return age;
18    }
19
20    public void setAge(int age) {
21        this.age = age;
22    }
23
24    public Person(String name, int age) {
25        super();
26        this.name = name;
27        this.age = age;
28    }
29
30 }
31
```

La plantilla example.html habría que modificarla así para poder identificar los datos que queremos mostrar:

```
ExampleController.java  example.html  ⌕
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8"/>
5 <title>HELLO</title>
6 </head>
7 <body>
8 <h1>Hello <span th:text="${person.name}"></span></h1>
9 <h3>Your age is: <span th:text="${person.age}"></span></h3>
10 </body>
11 </html>
```

Y en el controlador realizaríamos las siguientes modificaciones para poder usar los datos de esa clase modelo:

```

3 import org.springframework.stereotype.Controller;
10
11 @Controller
12 @RequestMapping("/example")
13 public class ExampleController {
14
15     public static final String EXAMPLE_VIEW = "example";
16
17     //Primera forma
18     @GetMapping("/exampleString")
19     public String exampleString(Model model){
20         model.addAttribute("person", new Person("Jon", 23));
21         return EXAMPLE_VIEW;
22     }
23
24     //Segunda forma
25     @GetMapping("/exampleMAV")
26     public ModelAndView exampleMAV(){
27         ModelAndView mav = new ModelAndView(EXAMPLE_VIEW);
28         mav.addObject("person", new Person("Mikel", 30));
29         return mav;
30     }
31 }
32
33
```

Insertando listados en plantillas

Ahora vamos a simular que estamos recogiendo un listado de datos complejos y lo vamos a mostrar por la plantilla.

Lo que habría que modificar en el controlador sería lo siguiente:

```
13
14 @Controller
15 @RequestMapping("/example")
16 public class ExampleController {
17
18     public static final String EXAMPLE_VIEW = "example";
19
20     //Primera forma
21     @GetMapping("/exampleString")
22     public String exampleString(Model model){
23         model.addAttribute("people", getPeople());
24         return EXAMPLE_VIEW;
25     }
26
27     //Segunda forma
28     @GetMapping("/exampleMAV")
29     public ModelAndView exampleMAV(){
30         ModelAndView mav = new ModelAndView(EXAMPLE_VIEW);
31         mav.addObject("people", getPeople());
32         return mav;
33     }
34
35     private List<Person> getPeople(){
36         List<Person> people = new ArrayList<>();
37         people.add(new Person("Jon", 23));
38         people.add(new Person("Mikel", 30));
39         people.add(new Person("Eva", 43));
40         people.add(new Person("Peter", 18));
41         return people;
42     }
43
44 }
```

La plantilla para recoger esos datos enviados por el controlador se modificaría y quedaría de la siguiente manera utilizando las propiedades thymeleaf.

Se mostraría una tabla con todos los datos recibidos desde el controlador.


```

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1"/>
<title>Insert title here</title>
</head>
<body>
    <table>
        <thead>
            <tr>
                <th>NAME</th>
                <th>AGE</th>
            </tr>
        </thead>
        <tbody th:each="person : ${people}">
            <tr>
                <th th:text="${person.name}"></th>
                <th th:text="${person.age}"></th>
            </tr>
        </tbody>
    </table>
</body>
</html>

```

Recibiendo una petición GET (1ª Forma)

Creamos una plantilla example2.html dentro de la carpeta templates.

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="ISO-8859-1"/>
    <title>Insert title here</title>
</head>
<body>
    <h1> Hello <span th:text="${nm_in_model}"></span></h1>
</body>
</html>

```

En nuestro paquete controller creamos una clase Example2Controller.java con el siguiente código:

```

@Controller
@RequestMapping("/example2")
public class Example2Controller {

    public static final String EXAMPLE2_VIEW="example2";

    @GetMapping("/request1")
    public ModelAndView request1(@RequestParam(name="nm", required=false,
defaultValue="NULL") String name) {
        ModelAndView mav=new ModelAndView(EXAMPLE2_VIEW);
        mav.addObject("nm_in_model", name );
    }
}

```



```

        return mav;
    }
}

```

Si ponemos en nuestro navegador cualquiera de las siguientes rutas, el controlador recuperará los datos enviados mediante GET en la ruta del navegador y los enviará a la plantilla para mostrarlos en el atributo de thymeleaf.

localhost:8080/example2/request1?nm=JON
localhost:8080/example2/request1?nm=MIKEL

Recibiendo una petición GET (2ª forma)

Dejamos la misma plantilla example2.html y modificamos la clase Example2Controller.java de la siguiente manera:

```

@Controller
@RequestMapping("/example2")
public class Example2Controller {

    public static final String EXAMPLE2_VIEW="example2";

    @GetMapping("/request2/{nm}")
    public ModelAndView request2(@PathVariable("nm") String name) {
        ModelAndView mav=new ModelAndView(EXAMPLE2_VIEW);
        mav.addObject("nm_in_model", name );
        return mav;
    }
}

```

Ahora en el navegador pondríamos de la siguiente manera la ruta para enviar la petición GET.

localhost:8080/example2/request2/JON
localhost:8080/example2/request2/MIKEL

Recibiendo una petición POST

Creamos una plantilla llamada **form.html** en nuestra carpeta **templates** para un formulario que utilizaremos para introducir los datos.

```

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1"/>
<title>FORM</title>
</head>
<body>

```

```

        <form action="#" th:action="@{/example3/addperson}" th:object="${person}"
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>

```

Creemos la plantilla llamada result.html que mostrará el resultado de enviar los datos del formulario.

```

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1"/>
<title>RESULT</title>
</head>
<body>
    <h1> PERSON: <span th:text="${person.name}"/></span> AGE:
    <span th:text="${person.age}"/></span> ADDED!!!</h1>
</body>
</html>

```

Crearemos un controlador llamado **Example3Controller.java** que tendrá dos métodos, uno para mostrar el formulario para introducir datos indicándole que le enviamos el modelo de datos de la clase Persona, y el otro método se encargará de recoger el modelo de datos del objeto persona con los datos introducidos y lo añadirá a la vista de la plantilla result.html (para ello utiliza la anotación **@PostMapping**).

```

@Controller
@RequestMapping("/example3")
public class Example3Controller {

    public static final String FORM_VIEW="form";
    public static final String RESULT_VIEW="result";

    //localhost:8080/example3/showForm
    @GetMapping("/showForm")
    public String showForm(Model model) {
        model.addAttribute("person", new Person());
        return FORM_VIEW;
    }

    @PostMapping("/addperson")
    public ModelAndView addPerson(@ModelAttribute("person") Person person) {
        ModelAndView mav=new ModelAndView(RESULT_VIEW);
        mav.addObject("person", person);
        return mav;
    }
}

```

Redirecciones

Para redireccionar nuestras plantillas si no indicamos en el `@GetMapping` la ruta específica de la dirección existen dos formas de hacerlo:

```
1ª Forma
@GetMapping("/")
public String redirect() {
    return "redirect:/example/showForm";
}

//2ª Forma
@GetMapping("/")
public RedirectView redirect() {
    return new RedirectView("/example3/showForm");
}
```

Página 404. NOT FOUND.

En nuestro fichero de configuración `application.yml` podemos poner la siguiente propiedad que viene en el appendix A, para dejar por defecto la página de tomcat cuando metamos una url incorrecta.

```
server:
  error:
    whitelabel:
      enabled: false
```

Pero si lo que queremos es que nos muestre la página 404 de error que nosotros tenemos personalizada tendremos que hacer lo siguiente:

Crear una carpeta `error` dentro de la carpeta `templates` y ahí metemos nuestra página personalizada de error 404.

Página 500. Internal Server Error

Si se produce alguna excepción o error no controlado para que nos muestre esta página personalizada se hace igual que antes. En la carpeta `error` dentro de la carpeta `templates` metemos nuestra página personalizada de Internal Server Error 500.

Creamos una clase en el controlador para controlar las excepciones Java que se produzcan.

```
@ControllerAdvice //Cuando se produce un error String vendrá a esta clase
public class ErrorController {

    public static final String ISE_VIEW="error/500";

    @ExceptionHandler(Exception.class) //Se capturan todos los errores de java
    public String showInternalServerError() {
```

```

        return ISE_VIEW;
    }
}

```

LOG para nuestra aplicación

Utilizamos la clase Log de la librería org.apache.commons.logging.

Vamos a añadir a nuestra clase Example3Controller.java una serie de Logs para probar como funcionan:

```

@Controller
@RequestMapping("/example3")
public class Example3Controller {

    private static final org.apache.commons.logging.Log
    LOGGER=LogFactory.getLog(Example3Controller.class);

    public static final String FORM_VIEW="form";
    public static final String RESULT_VIEW="result";

    @GetMapping("/")
    public RedirectView redirect() {
        return new RedirectView("/example3/showForm");
    }

    @GetMapping("/showForm")
    public String showForm(Model model) {
        LOGGER.info("Info Trace");
        LOGGER.warn("WARNING TRACE");
        LOGGER.error("Error Trace");
        LOGGER.debug("Debug Trace");
        model.addAttribute("person", new Person());
        return FORM_VIEW;
    }

    @PostMapping("/addperson")
    public ModelAndView addPerson(@ModelAttribute("person") Person person) {
        LOGGER.info("METHOD: 'addPerson' -- PARAMS: '"+person+"'");
        ModelAndView mav=new ModelAndView(RESULT_VIEW);
        mav.addObject("person", person);
        LOGGER.info("TEMPLATE: "+RESULT_VIEW+ " ---DATA: '"+person+"'");
        return mav;
    }
}

```

Y el resultado que obtendríamos por consola al llamar al método de la ruta **showForm** y luego al de **addPerson** sería:

```

2017-08-02 08:55:21.939 INFO 5692 --- [nio-8080-exec-1]
com.udemy.controller.Example3Controller : Info Trace

```

```
2017-08-02 08:55:21.939 WARN 5692 --- [nio-8080-exec-1]
com.udemy.controller.Example3Controller : WARNING TRACE
2017-08-02 08:55:21.939 ERROR 5692 --- [nio-8080-exec-1]
com.udemy.controller.Example3Controller : Error Trace
2017-08-02 08:55:30.738 INFO 5692 --- [nio-8080-exec-2]
com.udemy.controller.Example3Controller : METHOD: 'addPerson' -- PARAMS: 'Person
[name=John, age=33]'
2017-08-02 08:55:30.738 INFO 5692 --- [nio-8080-exec-2]
com.udemy.controller.Example3Controller : TEMPLATE: result---DATA: 'Person
[name=John, age=33]'
```

Expresiones Thymeleaf avanzadas

Para trabajar con Thymeleaf al completo tenemos el siguiente enlace:

<http://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#introducing-thymeleaf>