0ª Emisión

DIPLOMADO

Desarrollo de Sistemas con Tecnología Java

Módulo 8

Desarrollo de aplicaciones Web con Spring Web MVC

Ing. Jorge Alberto Montalvo Olvera

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Dirección General de Cómputo y de Tecnologías de información y Comunicación
Dirección de Docencia en TIC

DGTIC

Educación Continua
1971 - 2021

TEMARIO

- Spring Framework
- Spring Web MVC
  - DispatcherServlet

Spring Framework

- Spring es un Framework Java diseñado para agilizar el desarrollo de aplicaciones empresariales, este implementa una contenedor de inversión de control (IoC), puede ser usado para la programación aplicaciones web o de escritorio estándar, cuenta con módulos para: acceso a datos con JDBC, ORM, JPA, etc., crear aplicaciones web MVC, entres otros.

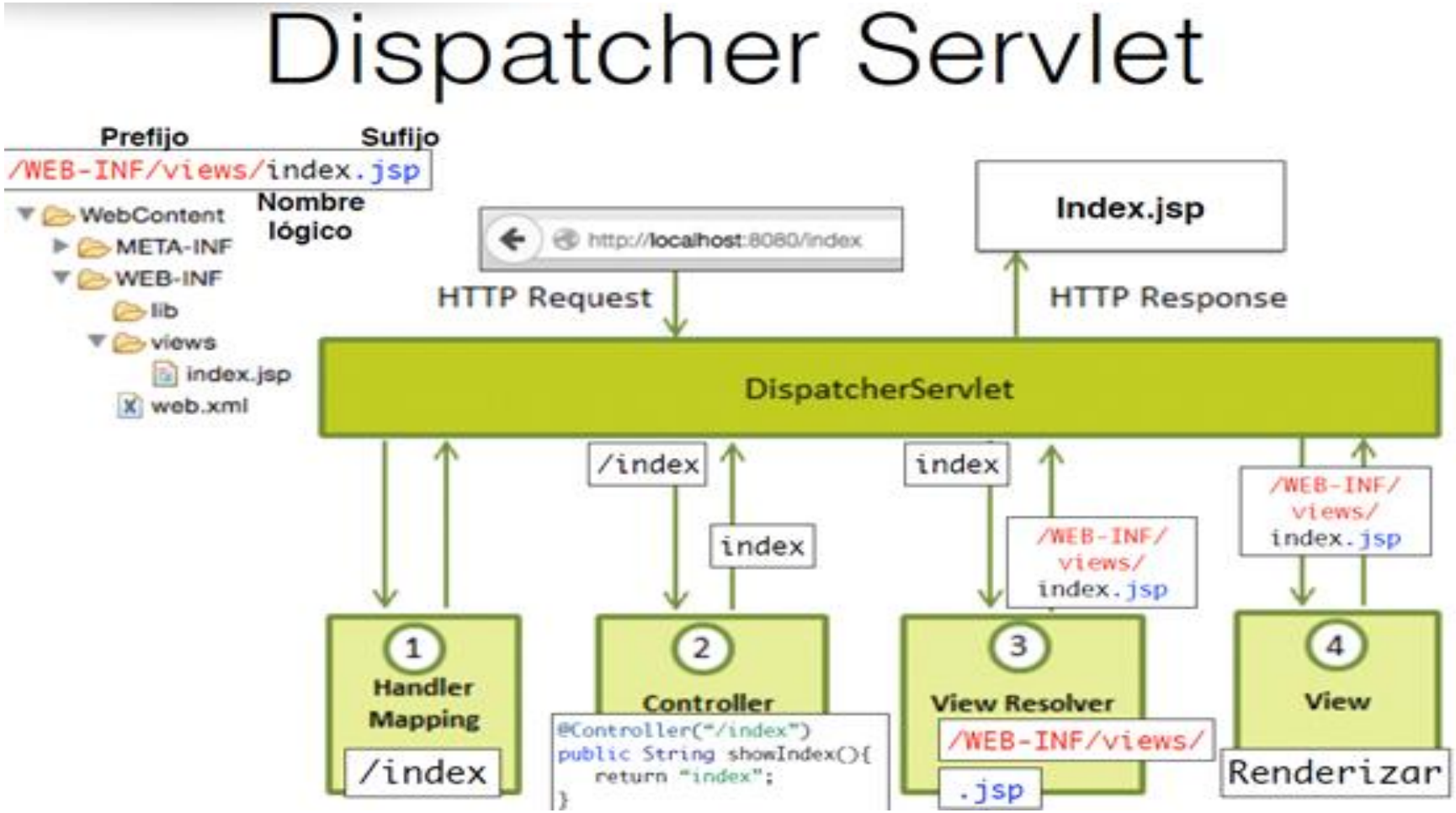
Spring Web MVC

- Spring Web MVC es un sub-proyecto Spring que esta dirigido a optimizar el proceso creación de aplicaciones web utilizando el patrón Modelo-Vista-Controlador.
  - Modelo → datos o información que manejará la aplicación web
  - Vista → elementos de la UI (Interfaz de Usuario)
  - Controlador → encargado manipular los datos en base a la interacción del usuario.

Distpacher Servlet

- La pieza central de Framework Spring MVC es el DispatcherServlet que extiende la clase HttpServlet este componente es el encargado de recibir las peticiones HTTP y generar la respuesta adecuada a dicha petición, se configura utilizando el archivo web.xml

Distpacher Servlet



Distpacher Servlet

- Handler mapping detecta al petición /index y retorna al contenedor buscando un @Controller que responda.
- @Controller devolverá el nombre lógico de la vista “index”.
- El contenedor envía la respuesta a un componente View Resolver que mediante prefijo y sufijo resuelve la ruta completa de la vista.
- Al final un componente View renderizará el resultado final.

Controllers

- Un controlador es el encargado de preparar el modelo (los datos manejados por la aplicación) y seleccionar el nombre de la vista que será utilizada para mostrar el modelo al cliente.
- El modelo es una implementación de la interface Map en la cual podemos almacenar datos enviados a la vista.
- Un controlador es capaz de generar una respuesta sin necesidad de una vista, esto es útil a la hora de crear servicios que generan respuestas en formatos como: XML, JSON, etc.

Controllers

- Los métodos encargados recibir la petición y resolver el nombre lógico, deben estar anotados con @RequestMapping para definir la petición HTTP.

```
@Controller
public class IndexController {

    @RequestMapping("/")
    public String showIndex() {
        return "index";
    }

    @RequestMapping("/about")
    public String showAbout() {
        return "about";
    }
}
```

View Resolver

- Las clases que implementen la interface ViewResolver serán las encargadas de resolver la vista, es decir, obtener el archivo físico que se usará para generar la vistas a partir del nombre lógico devuelto por el controlador.

  – InternalResourceViewResolver
  – JasperReportsViewResolver
  – ThymeleafViewResolver

View Model

- ModelAndView contiene los datos del modelo y el nombre de la vista.
  - ModelAndView mv = new ModelAndView();
  - mv.setViewName("hello");
  - return mv;
- Model rellena el modelo a través de la interface y retorna el nombre de la vista.
  - return "hello";

Configuracion

- @Configuration
  - Anotación que nos permite indicar que una clase en Spring MVC será quien representa la configuración de nuestra aplicación

- @Bean
  - Anotación que nos permite definir un bean dentro de nuestra aplicación en Spring MVC

Configuración

- @ComponentScan
  - Si el contenedor Spring lo creamos o configuramos mediante código Java con la clase AnnotationConfigApplicationContext, usamos la anotación @ComponentScan

- @EnableWebMvc
  - Anotación utilizada para activar el módulo Spring MVC y poder aplicar la configuración definida.
    - WebMvcConfigurer para personalización de Spring MVCConfiguración.

Patrones de plantilla URI

- Las plantillas URI se utilizan para facilitar el acceso a determinadas partes de una URL, en un método @RequestMapping.
- Una plantilla URI es una cadena que contiene uno o más nombres de variables. Al sustituir los valores de estas variables, la plantilla se convierte en un URI. Las plantillas URI están definidas cómo un URI parametrizado.
  - http://www.example.com/users/{userId} contiene la variable userId.
  - @PathVariable se usa como un argumento de un método para hacer que tome el valor de una variable de la plantilla URI.

Método de Petición

- RequestMethod
  - Permite indicar el método de petición HTTP con el que invoca la URI definida en el controller.
    - RequestMethod.GET
    - RequestMethod.POST

Parámetros en Petición

- @RequestParam
  - Anotación para el acceso a los parámetros específicos de la petición Servlet. (@RequestParam("petId") int petId).
  - Se puede indicar si el parámetro es obligatorio o no, así como definir un valor por default.

Recursos Estáticos

- Los recursos estáticos deben ser configurados para que podamos tener acceso a ellos.
  - Imágenes, archivos CSS o JavaScript, archivos multimedia como audio o video.

- ResourceHandlerRegistry
  - Permite registrar la carpeta que contendrá los recursos estáticos así como asignar un alias a dicha carpeta dentro de la configuración
    - registry.addResourceHandler("/resources/**")
      .addResourceLocations("/resources/")

Atributos de Modelo

- @ModelAttribute
  - Anotación que se puede utilizar a nivel de método. Indica que el propósito de ese método consiste en agregar uno o más atributos al modelo.
    - @ModelAttribute("productosList")
    - public List<Producto> getProductosList() {

Formatos View

- AbstractPdfView
  - Clase abstracta que permite dar una respuesta de tipo PDF
- AbstractXlsView
  - Clase abstracta que permite dar una respuesta de tipo xls
- MappingJackson2JsonView
  - Clase que permite dar como respuesta datos en formato json

Etiquetas Spring

- Spring define sus propias librerías de tags de forma similar a las ofrecidas por JSTL. Pero su utilización es opcional, no se exige su uso.
  - <%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
  - <%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>

Etiquetas Spring

- spring:message
  - Este tag proporciona el texto de un mensaje a partir de su código, resuelto en base a la configuración de internacionalización
    - <spring:message code='saludo’/>
- spring:url
  - Este tag permite generar un enlace del tipo a href de la misma manera que la etiqueta c:url de jstl.

Etiquetas Spring

- form:form
  - Este tag contiene a todos los demás tags de formularios, ofreciendo sobre todo facilidades para el binding.
- form:input
  - Este tag representa un campo de tipo input text.

Etiquetas Spring

- Controles Dentro del tag de formulario se pueden incluir los siguientes controles:
  - form:checkbox
  - form:checkboxes
  - form:radiobutton
  - form:radiobuttons
  - form:password
  - form:select
  - form:option
  - form:options
  - form:textarea
  - form:hidden
  - form:errors

Validaciones

- Para poder realizar la validación en formularios utilizando Spring MVC es posible realizarlo mediante 2 Apis.
  - javaee-web-api
    - @Size(min=3, max=20)
    - @Pattern(regexp="...")
    - @NotNull
    - @Min(18)
  - Hibernate validator
    - @NotBlank
    - @Email

Validaciones

- @Valid
  - Anotación encargada de validar los datos

- BindingResult
  - Nos permite verificar si hubo o no un error en la validación de los datos

Cache

- Spring aporta soluciones de Cache que permiten almacenar en memoria datos.
  - response.addHeader("Cache-Control", “no-cache");
  - setCacheControl(CacheControl.maxAge(60, TimeUnit.SECONDS)
    .noTransform().mustRevalidate())

Contacto

Jorge Alberto Montalvo Olvera

Ingeniero en Computación

jorge.Montalvo@gm3s.com.mx