FRAMEWORK

Spring MVC

Módulo II

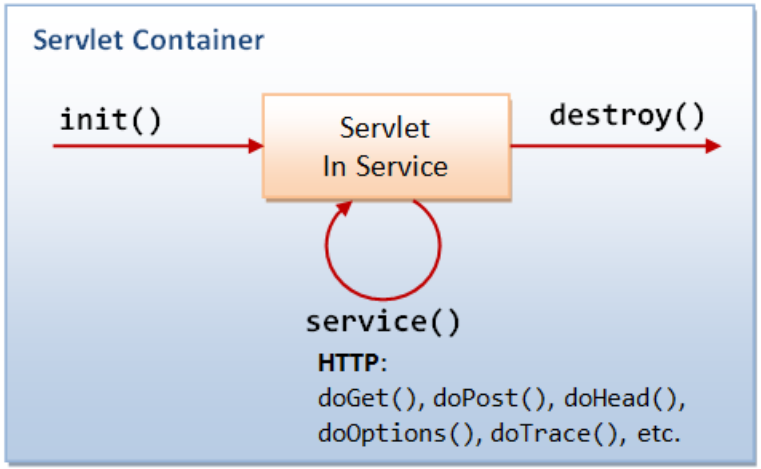
harriague+asociados

Emilio Rey | María Agustina Aliaga

INCLUIT
empowering your business

Servlet

Es una clase java que extiende de un HttpServlet basada en el modelo Cliente-Servidor, estos utilizan el protocolo Http como medio de comunicación y sirve para crear web dinámicas.



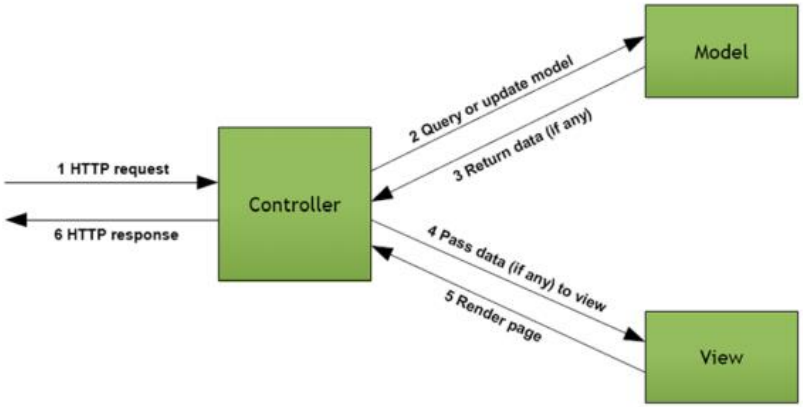

```
public class SimpleServlet extends HttpServlet {

    // Maneja el método GET de HTPP para
    // construir una sencilla página Web.

    public void doGet (HttpServletRequest request, HttpServletResponse response)
              throws ServletException, IOException {
    PrintWriter out;
    String title = "Simple Servlet Output";

    // primero selecciona el tipo de contenidos y otros campos de cabecera de la respuesta
    response.setContentType("text/html");
    // Luego escribe los datos de la respuesta
    out = response.getWriter();
    out.println("<HTML><HEAD><TITLE>");
    out.println(title);
    out.println("</TITLE></HEAD><BODY>");
    out.println("<H1>" + title + "</H1>");
    out.println("<P>This is output from SimpleServlet.");
    out.println("</BODY></HTML>");
    out.close();
    }
}
```

¿Que necesito para deployar un Servlet?

- JDK
- Container Web(puede ser un tomcat, jboss, weblogic, … etc)
- Definir el descriptor web(web.xml)
- servlet-api.jar

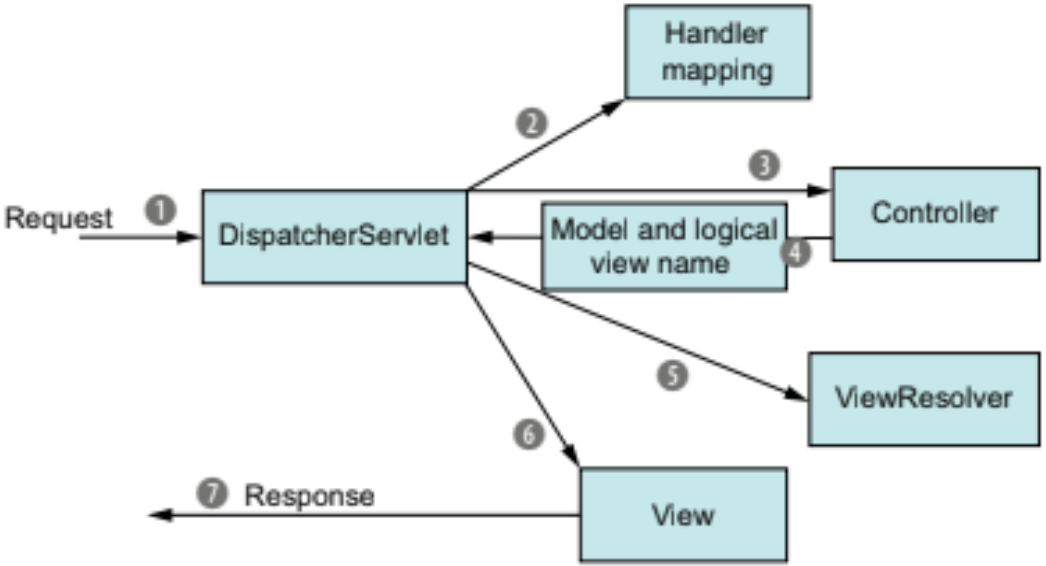

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://ja
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://j
    id="WebApp_ID" version="2.5">
    <display-name>library-web</display-name>
    <servlet>
        <servlet-name>CreateUserServlet</servlet-name>
        <servlet-class>library.servlets.CreateUser</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>CreateUserServlet</servlet-name>
        <url-pattern>/create_user</url-pattern>
    </servlet-mapping>
</web-app>
```

Patrón MVC



Este patrón es base para la mayoría de las aplicaciones Web, luego cambian dependiendo de las implementaciones de cada framework y de cómo se implementen la aplicación y las view.

Ciclo de vida del Request con Spring



Ciclo de vida del Request con Spring

La primer parada del request al igual que muchos frameworks java, está a cargo de un front controller que direcciona los request a través de un simple front controller. Un front controller es un patrón usado en aplicaciones web, donde un simple servlet delega la responsabilidad a un componente para procesar y resolver el request, en el caso de Spring MVC el front controller es el **1)DispatcherServlet.** En este caso el DispatcherServlet es el encargado de enviar los request a un Controller específico.
Un **Controller** es un componente de spring que procesa un request.Pero una típica aplicación tiene más de un controller para recibir requests aquí es donde el DispatcherServlet necesita de uno o más **2)handler mapping** para saber dónde direccionar la próxima parada del request. Una vez que el controller apropiado es elegido el DispatcherServlet envía el request a este controller.
3)El **Controller** es el inicio o el punto de entrada a nuestra aplicación con la información del request, este debería hacer muy poco si fue bien diseñado y debería delegar responsabilidades para la lógica de negocio a uno o más servicios. Luego debería mostrarle la información al user en un browser, en este caso la información es lo que llama **model**. **4)**Una de las últimas cosas que el controller hace es empaquetar los datos del modelo e identificar la vista que debería renderizar la salida y envía el request con el modelo y nombre de la vista al **DispatcherServlet. 5)**El **DispatcherServlet** consulta un **ViewResolver** para mapear el nombre de la vista para una específica llamada a la implementación de esa vista, que puede ser o no un jsp.**6)**Ahora que el **DispatcherServlet** conoce que **vista** debería renderizar el resultado, el trabajo del request está casi cubierto. La última parada es la implementación de la vista, típicamente suele ser un jsp, donde se dejan los datos del modelo, hasta aquí llega el trabajo del request.
7)La vista usa los datos del modelo para renderizar la salida que debe ser mostrada al cliente con el response object.

Preparando nuestro entorno de desarrollo

- Instalar el **JDK**(Kit de desarrollo para java)
- Instalar la versión de **tomcat** correspondiente para el **JDK** que se instaló
- Necesitaremos un **IDE**(Entorno de desarrollo Integrado) que puede ser cualquiera para java, las más conocidas y más usadas son IntelliJ y Eclipse
- Importar el proyecto que se creó en el módulo 1 desde el ide.
- Comenzar a modificar el archivo **web.xml**
- Agregar el archivo **application-context.xml** al proyecto
- Agregar las librerias necesarias en el archivo **pom.xml**
  - **junit**
  - **spring-core**
  - **spring-webmvc**
  - **spring-context**

Archivos para crear y/o modificar

1. **springapp/src/main/webapp/WEB-INF/web.xml**
2. **springapp/src/main/webapp/WEB-INF/spring/application-context.xml**
3. **springapp/src/main/java/com/companyname/springapp/web/HelloController.java**
4. **springapp/src/test/java/com/companyname/springapp/web/HelloControllerTests.java**
5. **springapp/src/main/webapp/hello.jsp**

Agregar el servlet al archivo descriptor

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:web="http://java.sun.com/xml/ns/j2ee/web-app_2_5.xsd"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/j2ee/web-app_2_5.xsd">

 <display-name>Springapp</display-name>

 <servlet>
                <servlet-name>springapp</servlet-name>
                <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
                <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/spring/application-context.xml</param-value>
                </init-param>
                <load-on-startup>1</load-on-startup>
 </servlet>
```

```
  <servlet-mapping>
                <servlet-name>springapp</servlet-name>
                <url-pattern>*.htm</url-pattern>
  </servlet-mapping>

</web-app>
```

Agregar el archivo para cargar el contexto de spring

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:context="http://www.springframework.org/schema/context"
                xmlns:mvc="http://www.springframework.org/schema/mvc"
                xsi:schemaLocation="http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-beans.xsd
                http://www.springframework.org/schema/context
                http://www.springframework.org/schema/context/spring-context.xsd
                http://www.springframework.org/schema/mvc
                http://www.springframework.org/schema/mvc/spring-mvc.xsd">

                <!-- Scans the classpath of this application for @Components to deploy as beans -->
                <context:component-scan base-package="com.companyname.springapp.web" />

                <!-- Configures the @Controller programming model -->
                <mvc:annotation-driven/>

</beans>
```

Anotaciones

Annotation	Funcionalidad
@Controller	Esto marca como Controller a una clase, entonces el DispatcherServlet conoce esto y lo empieza a considerar para delegarle el request.
@RequestMapping	Esto nos indica cómo acceder al controller correspondiente a través de la interpretación de las urls.

Spring MVC provee varias maneras para que un cliente pueda pasar datos a un controller.
Estos son **QueryParameters, FormParameters, PathVariable.**

Annotation	Funcionalidad
@RequestParam	Esta anotación es la que relaciona a los parámetros de las urls(QueryParameters) con nuestra aplicación. Ejemplo: /algo?<param1>=value1&<param...>=value… Hay que tener en consideración que el nombre del parámetro tiene estar indicado dentro de la annotation. Ejemplo: /algo?max=5 @RequestParam(“max”) Integer maximo

Anotaciones

Annotation	Funcionalidad
@PathVariable	El **path variable** se refiere a cuando la url tiene una o más partes del path variable. Ejemplo: **/auto/{marca}/modelo/{año}** quedando la url **/auto/volvo/modelo/89** Para obtener el valor en nuestra aplicación hay que tener dos consideraciones: 1. La annotation @RequestMapping tiene que representar la url o path @Requestmapping(value="**/auto/{marca}/modelo/{anio}**", method="GET") 1. El mapeo con la variable para luego ser usada en nuestra aplicacion @PathVariable("marca") String marca, @PathVariable("anio") Integer anio

	FormParameters, esto es para cuando se completa un formulario y esos datos tienen que ser recibidos en el controller.
	La consideración para este caso es que el @RequestMapping tiene que ser un POST y el metodo debería contener un Objeto como parámetro.
	
	Ejemplo:
	@RequestMapping(value="/usuario" method="POST")
	public String guardarUsuario(Usuario usuario) {
	.
	.
	}

Controller

```
@Controller
public class HelloController {

    protected final Log logger = LogFactory.getLog(getClass());

    @RequestMapping(value="/hello.htm")
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        logger.info("Returning hello view");

        return new ModelAndView("hello.jsp");
    }
}
```

Test

```
package com.companyname.springapp.web;
import static org.junit.Assert.*;
import org.junit.Test;
import org.springframework.web.servlet.ModelAndView;

public class HelloControllerTests {

@Test
   public void testHandleRequestView() throws Exception{
      HelloController controller = new HelloController();
      ModelAndView modelAndView = controller.handleRequest(null, null);
      assertEquals("hello.jsp", modelAndView.getViewName());
   }
}
```

Vista

```
<html>
   <head><title>Hello :: Spring Application</title></head>
   <body>
      <h1>Hello - Spring Application</h1>
      <p>Greetings.</p>
   </body>
</html>
```