

Spring MVC

Spring MVC	1
¿Qué aprenderás?	2
Introducción	2
Vista	2
Controlador	2
Front Controller	3
Herramientas proporcionadas para la Vista	6
JSTL	6
Agregar JSTL a nuestro proyecto	6
Agregar JSTL a nuestra vista (jsp)	7
Funciones de JSTL	7
Spring MVC Form	9



¡Comencemos!

¿Qué aprenderás?

- Configurar e implementar el dispatcher de spring en la capa del Front Controller.
- Implementar sentencia JSTL y Forms a las vistas JSP.

Introducción

Spring MVC es una herramienta integrada en el framework, la cual permite desarrollar el software bajo la arquitectura que plantea el patrón de diseño Modelo Vista Controlador. Esta contiene variadas herramientas que facilitan el desarrollo de software a los programadores, ya que muchas de estas ofrecen soporte a implementaciones tanto para la capa Vista como para el Controlador. A continuación, veremos estas herramientas.

¡Vamos con todo!



Vista

- **Custom Tags JSTL:** Implementada mediante la librería JSTL. Con esta podemos introducir código java en nuestras páginas HTML que será interpretado.
- **Custom Tags Form:** Integrada en el framework de Spring. Esta nos facilita la integración de formularios que se comuniquen con nuestro controlador.

Controlador

- **RequestMapping:** Anotación que permite al controlador marcar las acciones e identificar las mismas para ser usadas por la vista.
- **RequestParam:** Anotación que permite identificar los parámetros recibidos desde la vista, que necesitan una acción en el controlador.
- **ModelAttribute:** Anotación que permite identificar un modelo utilizado en la vista.

- **ModelMap:** Objeto que permite identificar las distintas variables del modelo junto al valor de cada una de estas, de tal forma que la vista pueda hacer uso de dichas variables.
- Características de Spring MVC.

Dentro de las características más importantes de Spring MVC, encontramos las siguientes:

- Permite separar las distintas capas de la arquitectura MVC.
- Los componentes de Spring MVC son fácilmente testeables, aumentando la eficacia y robustez de los desarrollos.

En resumen, Spring Framework, y particularmente Spring MVC hace más fácil la vida de los desarrolladores, permitiéndoles enfocar su atención en la lógica, más que en las configuraciones tediosas y funciones transversales al sistema.

Front Controller

El Front Controller, es un patrón de diseño que se basa en un solo controlador como punto de entrada a la aplicación, el cual se encarga de recibir y gestionar todas las peticiones de los usuarios o clientes. Algunas de las funciones más comunes del Front Controller, son las siguientes:

- Comprobación de los aspectos de seguridad del sistema, por ejemplo, comprobar que las peticiones de los clientes sean bajo el protocolo HTTPS.
- Mapeo y delegación de las peticiones hacia los controladores específicos de Spring, los cuales se encargan de generar la vista adecuada a los usuarios.
- Manejo de errores.

El Front Controller, necesita de un objeto Dispatcher (Despachador), el cual se encarga de redireccionar todas las peticiones de los usuarios, que ha aceptado el Front Controller, a los objetos controladores de Spring.

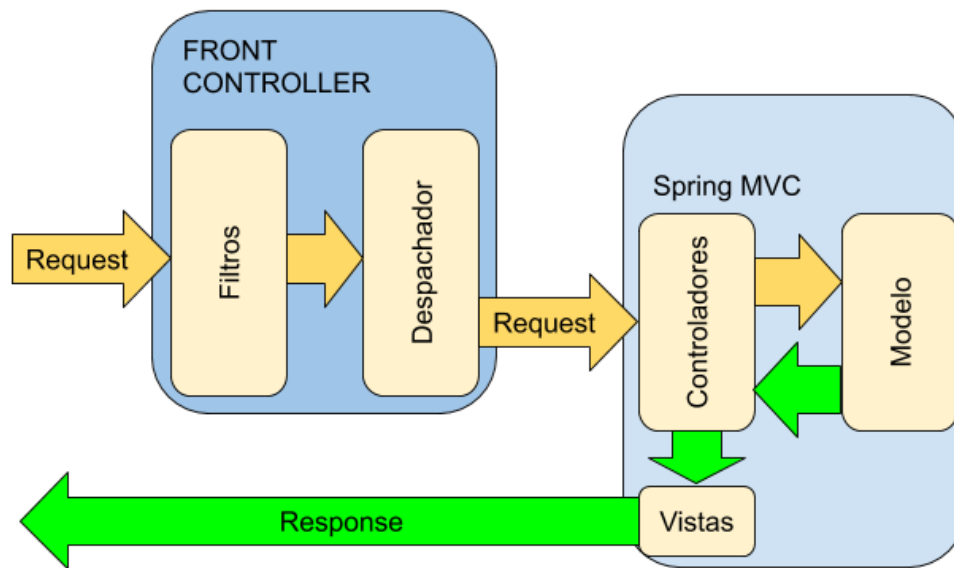


Imagen 1. Petición request y su respuesta response.
Fuente: Desafío Latam.

En la imagen 1, se muestra el flujo completo de una petición (Request) y su respuesta (Response), considerando el Front Controller y SPRING MVC. El usuario realiza una petición, entrando directamente y siempre en primera instancia, al Front Controller. Implementa los filtros necesarios y previamente configurados, para evaluar si la petición aplica para delegarla al despachador. Una vez que el Front Controller evalúa y aprueba la petición, se la delega al despachador, quien tiene la responsabilidad de buscar el controlador específico para dicha petición. Luego, el controlador específico le pide al modelo los datos solicitados, el modelo responde con los datos al controlador, y este último genera la vista con los datos requeridos por el usuario.

Para configurar el Front Controller, es necesario incluir en el proyecto un archivo xml llamado web.xml, el cual debe estar alojado bajo el directorio WEB-INF del proyecto. Un ejemplo de este archivo es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app metadata-complete="true"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
id="WebApp_ID"
version="3.0">
<display-name>ContactManager</display-name>
<servlet>
<servlet-name>spring</servlet-name>
```

```
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>*.html</url-pattern>
    </servlet-mapping>
</web-app>
```

En la primera parte, se configura el Despachador que provee Spring, definido por la clase `org.springframework.web.servlet.DispatcherServlet`, para que se encargue de buscar los controladores específicos en base a las peticiones entrantes.

```
<servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

En la segunda parte del archivo, se accede al servlet especificado con el nombre Spring, en este caso al dispatcher. Lo que se le está indicando entonces, es que todas las urls o peticiones entrantes deben ser o terminar con `.html`.

```
<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>*.html</url-pattern>
</servlet-mapping>
```

Herramientas proporcionadas para la Vista

Spring MVC, ofrece distintas herramientas que permiten, entre otras cosas, realizar operaciones en la Vista de nuestro sistema. Algunas de las herramientas más utilizadas, son las siguientes:

JSTL

Esta es una librería ampliamente utilizada en la vista, principalmente en los archivos jsp (Java Server Page), en donde se codifica la lógica que tendrá la vista. Dentro de las funciones más utilizadas de JSTL, están las siguientes:

- Sentencias condicionales (if, choose, when, otherwise).
- Sentencias iterativas (forEach).
- Visualización de variables del modelo (out).
- Asignación de valores al modelo (set).
- Otras.

Agregar JSTL a nuestro proyecto

JSTL, no viene integrada dentro del pool de librerías de Spring, por lo que debemos agregarla, recordando maven, dentro de las dependencias de nuestro archivo pom.xml, de la siguiente manera:

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
</dependency>
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
</dependency>
```

Agregar JSTL a nuestra vista (jsp)

Una vez que ya tenemos las librerías necesarias agregadas en nuestro archivo pom.xml del proyecto, debemos declarar el tag, haciendo referencia a las funciones de la librería, dentro de la vista jsp, para así poder hacer uso de las herramientas que la librería ofrece. La forma de hacerlo es la siguiente:

- En nuestro archivo jsp, al inicio de este se debe declarar el tag de la siguiente forma:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

El atributo prefix="c", indica a la vista que JSTL Core se utilizará bajo el prefijo c, por lo que los tag se usarán, por ejemplo de la siguiente manera: <c:out /> o bien <c:set />.

Funciones de JSTL

- **out:** Sirve para imprimir el valor de una variable.

```
<c:out value="${variable}" />
```

- **set:** Sirve para asignar un valor a una variable

```
<c:set var="edad" value="${18}"/>
```

En el ejemplo, se le está asignando a la variable edad el valor 18

- **if:** Sirve para evaluar una expresión:

```
<c:set var="edad" value="${18}"/>
<c:if test = "${edad > 17}">
  <p>Mayor de edad<p>
</c:if>
```

El atributo test, recibe la expresión a evaluar. En este caso, se está evaluando si la edad es mayor que 17.

```
<c:if test = "${edad > 17}" var = "result">
  <p>Mayor de edad<p>
</c:if>
```

- **choose, when, otherwise:** Sirven en conjunto para realizar varias condiciones, similar al if else de java.

```
<c:choose>
  <c:when test="${condicion1}">
    //Aquí van las líneas a ejecutarse en caso de que condicion1 se
    cumpla
  </c:when>
  <c:when test="${condicion2}">
    //Aquí van las líneas a ejecutarse en caso de que condicion2 se
    cumpla
  </c:when>
  <c:otherwise>
    /*Si ninguna de las condiciones c:when anteriores se ha cumplido, se
    ejecutará esta fracción de código.*/
  </c:otherwise>
</c:choose>
```

- **forEach:** Sirve para ejecutar una sentencia iterativa, tal como el for o forEach en java.

```
<c:forEach var="contador" begin="0" end="10">
  //Esto hace un for, contando desde 0 hasta 10
</c:forEach>
```

Esta sentencia itera sobre una lista ArrayList llamada listaMensajes, y cada valor de la lista la deja en el mensaje variable.

```
<c:forEach items="${listaMensajes}" var="mensaje">
  <c:out value="${mensaje}" />
</c:forEach>
```

Más adelante veremos de manera práctica cómo ir usando algunas de las funciones de JSTL.

Spring MVC Form

Spring MVC Form, corresponde a un tag de Spring MVC que nos permite manejar formularios en nuestra vista. Una de las características más importantes de este tag, es que nos permite utilizar el modelo de Spring devuelto por el controlador.

Este tag viene incluido dentro de las librerías de Spring Boot, por lo que no es necesario agregar de forma explícita dicha librería al archivo pom.xml. Para poder utilizar este tag, es necesario agregar la siguiente línea al inicio de la vista jsp:

```
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
```

El atributo prefix="form", indica a la vista que Spring MVC Form se utilizará bajo el prefijo form, por lo que los tag se usarán, por ejemplo de la siguiente manera: form:form.

```
<form:form id="frmAcciones" modelAttribute="contact"
action="/addContact">
    <!-- Aquí va el código html, jstl u otro que pinte en pantalla el
formulario-->
</form>
```

modelAttribute, describe el nombre del atributo del modelo que se utilizará en el controlador. Es decir, que el controlador recibirá por parámetro un objeto declarado como contact. Por ejemplo:

```
<form:form id="frmAcciones" modelAttribute="contact"
action="/addContact">
    <div>
        <fieldset>
            <legend>Mantenedor de Contactos</legend>
            <table border="1">
                <tr>
                    <td>Nombre: </td>
                    <td><input type="text" id="idNombre" name="nombre"/></td>
                    <td>Apellido Paterno: </td>
                    <td><input type="text" id="idApellidoPaterno"
name="apellidoPaterno"/></td>
                </tr>
                <tr>
```

```
<td>Dirección: </td>
<td><input type="text" id="idDireccion" name="direccion"/></td>
<td>Teléfono: </td>
<td><input type="text" id="idTelefono" name="telefono"/></td>
</tr>
<tr>
<td colspan="4">
<input type="button" value="Buscar" />
<input type="button" value="Agregar"
onclick="submitFormAction('/contactManager/addContact', 'post')"/>
<input type="button" value="Eliminar" />
</td>
</tr>
</table>

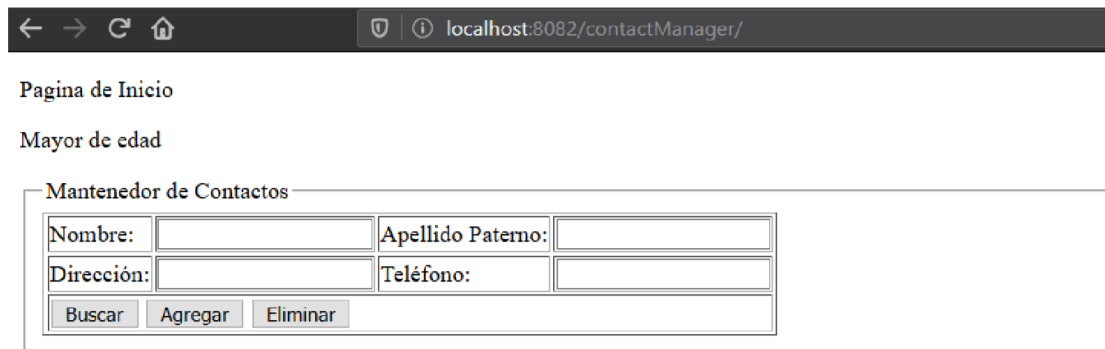
</fieldset>
</div>
</form:form>
```

En el ejemplo anterior, cada atributo name de cada objeto input de tipo texto, especifica un atributo del modelo contact.

Si implementamos ambos contenidos Spring MVC Form y JSTL podemos obtener lo siguiente:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form"
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Inicio</title>
</head>
<body>
<p>Pagina de <c:out value="${page_name}" /></p>
<c:set var="edad" value="${18}"/>
<c:if test = "${edad > 17}">
<p>Mayor de edad<p>
</c:if>
<form:form id="frmAcciones" modelAttribute="contact"
```

```
action="/addContact">
  <div>
    <fieldset>
      <legend>Mantenedor de Contactos</legend>
      <table border="1">
        <tr>
          <td>Nombre: </td>
          <td><input type="text" id="idNombre"
name="nombre"/></td>
          <td>Apellido Paterno: </td>
          <td><input type="text" id="idApellidoPaterno"
name="apellidoPaterno"/></td>
        </tr>
        <tr>
          <td>Dirección: </td>
          <td><input type="text" id="idDireccion"
name="direccion"/></td>
          <td>Teléfono: </td>
          <td><input type="text" id="idTelefono"
name="telefono"/></td>
        </tr>
        <tr>
          <td colspan="4">
            <input type="button" value="Buscar" />
            <input type="button" value="Agregar"
onclick="submitFormAction('/contactManager/addContact', 'post')"/>
            <input type="button" value="Eliminar" />
          </td>
        </tr>
      </table>
    </fieldset>
  </div>
</form:form>
</body>
</html>
```



The screenshot shows a web browser window with the address bar displaying 'localhost:8082/contactManager/'. The page content includes a header with navigation links 'Pagina de Inicio' and 'Mayor de edad'. Below this is a section titled 'Mantenedor de Contactos' which contains a form with four input fields: 'Nombre:', 'Apellido Paterno:', 'Dirección:', and 'Teléfono:'. At the bottom of the form are three buttons: 'Buscar', 'Agregar', and 'Eliminar'.

Nombre:	<input type="text"/>	Apellido Paterno:	<input type="text"/>
Dirección:	<input type="text"/>	Teléfono:	<input type="text"/>
<input type="button" value="Buscar"/> <input type="button" value="Agregar"/> <input type="button" value="Eliminar"/>			

Imagen 2. Visual en el navegador de Vista de Ejemplo.
Fuente: Desafío Latam.