Capítulo 5
El framework de presentación JSF

1. Visión general

El uso de servlets y JSP en un proyecto a gran escala, no resulta sencillo. Es preferible utilizar un framework que ofrezca un marco de arquitectura bien definido y un conjunto de componentes que limiten el código redundante e incómodo. El objetivo de este capítulo es descubrir los conceptos básicos del framework MVC **JSF 3.0** (*Jakarta Server Faces*), que es un framework orientado a componentes.

Un framework orientado a componentes es un framework que manipula los componentes visuales y de negocio, limitando al mismo tiempo el uso directo de las tecnologías web tanto como sea posible. En este sentido, este tipo de framework se parece al desarrollo de una aplicación cliente/servidor. Por lo tanto, los desarrolladores sin experiencia en el desarrollo web pueden obtener resultados rápidamente.

En oposición a este tipo de framework, existen otros orientados a la acción. Este tipo de framework maneja las peticiones y respuestas HTTP. Un framework de este tipo está mucho más cerca del protocolo HTTP y las tecnologías satelitales. Se basa claramente en las tecnologías presentadas en los capítulos anteriores. Por ejemplo, es el caso de Struts 2.

Sin embargo, ambos tipos de frameworks respetan el patrón MVC (modelo-vista-controlador). Por lo tanto, proponen una arquitectura que permite la separación entre el modelo, la vista y el controlador.

2. Presentación de JSF

2.1 Aspectos generales

JSF es una tecnología de Jakarta que actualmente se encuentra en su versión 3.0. La documentación oficial está disponible en:
https://jakarta.ee/specifications/faces/3.0/jakarta-faces-3.0.pdf

En este capítulo se utiliza la implementación de referencia, denominada **Mojarra 3.0**. El sitio web de referencia para esta implementación se puede encontrar en la dirección https://eclipse-ee4j.github.io/mojarra/

JSF consta de los siguientes elementos:

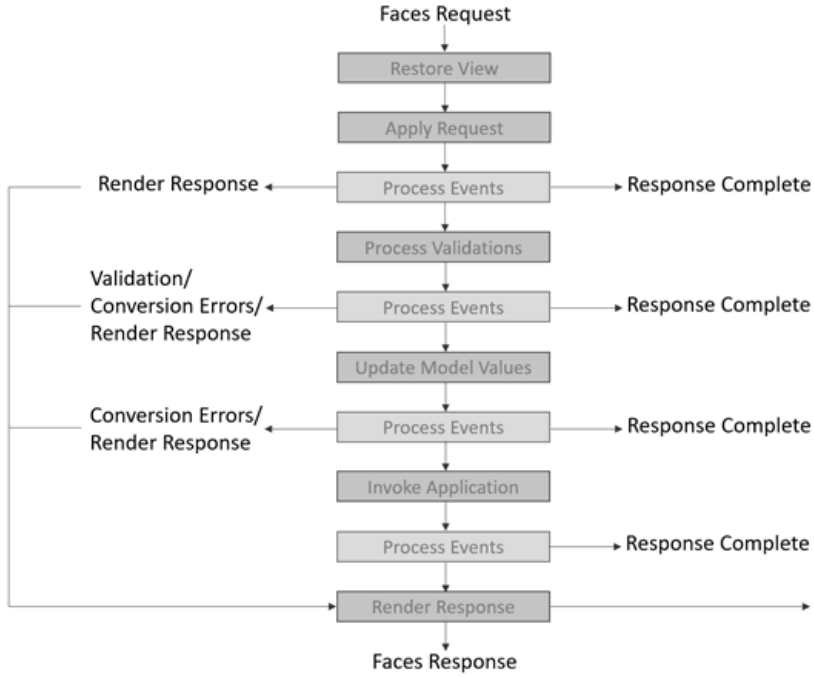
– Una API para representar componentes y administrar su estado. Estos componentes se denominan ***managedBeans*** (también llamados *backingBeans*).

– Una API para gestionar eventos, validaciones del lado del servidor, conversión de datos, navegación e internacionalización.

– Librerías de tags o etiquetas, como las disponibles para JSP (etiquetas JSTL), que permiten proporcionar una representación adecuada de *managedBeans*. Las páginas JSF se llaman facelets.

2.2 Principios de funcionamiento

El funcionamiento se basa en un conjunto de pasos (ciclo de vida) desde la llegada de la petición HTTP hasta la representación de la respuesta. El siguiente diagrama muestra cómo funciona esto.



El ciclo de vida se compone de seis etapas, entre las cuales se intercalan fases de gestión de eventos (*Process Events*) no cubiertas en este libro:

- ***Restore View***: esta es la primera etapa que se realiza cuando se ejecuta una petición a un recurso JSF. Esta etapa se utiliza para constituir en memoria un árbol de componentes (*components tree*), también llamado vista (*view*), que representa la página y los elementos manejados. Cuando se accede a la página por primera vez, simplemente se crea este árbol.
- ***Apply Request***: esta segunda etapa consiste en leer los parámetros de la petición para actualizar la vista con la información introducida por el usuario.
- ***Process Validations***: esta tercera etapa permite la conversión y validación de la información. La conversión permite pasar de una cadena de caracteres introducida por el usuario a un tipo adaptado al contexto de la petición. Este tipo puede ser clásico, como un numérico, una fecha, etc., pero también puede ser más complejo, como un objeto de tipo `Deporte`, `Superficie`, etc. En este caso, se requiere la participación de un conversor (*converter*) personalizado. Cuando la conversión se ha realizado correctamente, se realiza la validación. La validación implica el uso de validadores (*validator*) estándar o personalizados. Más adelante, en este capítulo, dedicamos una sección a cada una de estas dos actividades.

– ***Update Model Values***: esta cuarta etapa actualiza el modelo a partir de la vista. Como parte de un formulario de entrada de un nuevo deporte, si las entradas cumplen con las reglas de negocio, las propiedades de un objeto `Deporte` se valorizan con la información introducida por el usuario. Este objeto se encuentra en una clase Java denominada *managedBean*. Más adelante, en este capítulo, dedicamos una sección a *managedBeans*.
– ***Invoke Application***: esta quinta etapa es crucial porque desencadena el procesamiento de negocio relacionado con la petición HTTP. Este procesamiento se describe en un *managed Bean*, que actúa como un controlador.
– ***Render Response***: esta sexta y última etapa es la representación de la respuesta de la vista. Esta representación incluye los mensajes de error generados por las etapas previas.

Durante la primera petición a un recurso, solo se ejecutan los pasos *Restore View* y *Render Response*.

Estos diferentes pasos están controlados por un servlet, que sirve como punto de entrada a las diferentes peticiones. Este servlet se llama `jakarta.faces.webapp.FacesServlet`. Es el único servlet en un proyecto JSF. Actúa como un *Front Controller* porque su función es recibir peticiones HTTP entrantes, explotarlas y representar una respuesta adecuada. Las siguientes secciones están destinadas a presentar los elementos principales para comenzar con JSF.

2.3 El proyecto

El objetivo de este capítulo es comprender el funcionamiento básico de JSF a través de un proyecto de ejemplo llamado Proyecto_JSF.

- Empiece creando un **Gradle Project** con las mismas características que en el primer capítulo.
- Complete el árbol de directorios para que tenga el siguiente aspecto:

- Agregue un archivo denominado `web.xml` en el directorio `src/main/webapp/WEB-INF` con un contenido equivalente al proyecto inicial (*PrimerPROWeb*). Asegúrese simplemente de cambiar el nombre del proyecto dentro de la etiqueta `<display-name>`.
- Edite el archivo `build.gradle` con el siguiente contenido:

```
plugins {
   id 'java'
   id 'war'
}

repositories {
   jcenter()
}

dependencies {
//API Servlet y JSP para la compilación
compileOnly "jakarta.servlet:jakarta.servlet-api:5.0.0"
compileOnly group: 'jakarta.servlet.jsp', name:
'jakarta.servlet.jsp-api', version: '3.0.0'
//API JSTL para utilizar las etiquetas JSTL
implementation group: 'org.glassfish.web', name:
'jakarta.servlet.jsp.jstl', version: '2.0.0'
//API JDBC y JPA para utilizar una base de datos
/*implementation group: 'mysql', name: 'mysql-connector-java',
version: '8.0.26'
implementation group: 'org.hibernate', name:'hibernate-core-jakarta',
version:'5.6.2.Final'*/
//API JSF (Mojarra) para utilizar JSF
implementation 'org.glassfish:jakarta.faces:3.0.2'
//API CDI para utilizar la inyección de dependencias
implementation group: 'org.jboss.weld.servlet', name:
'weld-servlet-shaded', version: '4.0.2.Final'
//API Bean Validation para validar los datos
implementation 'org.hibernate.validator:hibernate-validator:7.0.2.Final'}
```

- Observe las nuevas dependencias:
  - La dependencia de **Mojarra 3.0.2** para usar JSF en el proyecto.
  - La dependencia de **Hibernate Validator 7.0.2.Final** para usar la especificación **Bean Validation** en el paso de la validación JSF.
- Seleccione el menú contextual **Gradle - Refresh Gradle Project** para que su proyecto descargue las dependencias.

El proyecto finalmente está listo.

3. Configuración general

3.1 El archivo faces-config.xml

El archivo `faces-config.xml`, ubicado en el directorio `webapp/WEB-INF`, es un archivo de configuración opcional en el uso de JSF. Debe tener el siguiente contenido mínimo, con la etiqueta raíz `<faces-config>`:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml version="1.0" encoding="UTF-8"?>
<faces-config
 xmlns="https://jakarta.ee/xml/ns/jakartaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
 https://jakarta.ee/xml/ns/jakartaee/web-facesconfig_3_0.xsd"
 version="3.0">

</faces-config>
```

A título informativo, este archivo solo se modifica ligeramente en el contexto de este capítulo porque los ejemplos implementados son relativamente simples y utilizan las anotaciones o configuraciones predeterminadas.

3.2 El archivo beans.xml

El archivo `beans.xml`, ubicado en el directorio `webapp/WEB-INF`, es un archivo de configuración necesaria para usar la inyección de dependencias a través de la API Weld. Debe tener el siguiente contenido mínimo con la etiqueta raíz `<beans>`:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="https://jakarta.ee/xml/ns/jakartaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/beans_3_0.xsd"
bean-discovery-mode="annotated">

</beans>
```

A título informativo, este archivo no se modificará en el ámbito de este capítulo porque los ejemplos implementados son relativamente simples y utilizan las anotaciones o configuraciones predeterminadas.

3.3 El archivo web.xml

El archivo `web.xml` que, como ya sabe, se encuentra en el directorio `webapp/WEB-INF`, debe tener al menos el siguiente contenido:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ... version="5.0">
 <display-name>Proyecto_JSF</display-name>
 <servlet>
 <servlet-name>Faces Servlet</servlet-name>
 <servlet-class>jakarta.faces.webapp.FacesServlet</servlet-class>
 <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
 <servlet-name>Faces Servlet</servlet-name>
 <url-pattern>*.xhtml</url-pattern>
 </servlet-mapping>
</web-app>
```

Contiene la declaración y configuración del servlet `FacesServlet`. Por lo tanto, el framework JSF procesará todas las URL que terminan en `.xhtml`. El servlet `FacesServlet` es el punto de entrada para el uso de este framework.

Este archivo se debe modificar, en el contexto del proyecto, para definir ciertos comportamientos que son deseables durante la ejecución de la aplicación. Estos cambios consisten en agregar parámetros de aplicación a través de la etiqueta `<context-param>`.

Los parámetros posibles son bastante numerosos. Una lista completa de parámetros está disponible en la documentación oficial de la especificación (sección Application Configuration Parameters). Como parte del proyecto, se utilizan tres parámetros:

- `jakarta.faces.STATE_SAVING_METHOD`: este parámetro define dónde se guarda el estado de la vista. Los valores posibles son `server` y `client`. El valor predeterminado es `server`. Si el valor es `server`, la información se guarda en el servidor, normalmente en `session`. Si el valor es `client`, es decir, información sobre el cliente, normalmente en un campo oculto en un formulario.
- `jakarta.faces.PROJECT_STAGE`: este parámetro permite tener un comportamiento adaptado en caso de problemas en tiempo de ejecución. En la fase de desarrollo, es interesante asignar a parámetro el valor `Development` para tener información detallada sobre el origen del error. Los valores posibles son `Development`, `UnitTest`, `SystemTest` y `Production`.