Introducción a JSF y Primefaces

ESEI Dojos 2012/13

30 de noviembre de 2012

Índice

1. Material

VirtualBOX: http://www.virtualbox.org/

Imagen máquina virtual: dojo-jsf.vdi.gz [comprimida, 400 MB]

Imagen SWAP: swap.vdi [20 KB]

Scripts de configuración

- GNU/Linux: dojo-jsf.sh

```
bash dojo_jsf.sh
```

- Windows: dojo-jsf.bat **Aviso:** ajustad la variable `VBOX_PATH` indicando el path de instalación de VirtualBOX

Usuarios configurados.

login	password
dojo	dojo
root	purple

1.1. Creación del proyecto Netbeans

1. Crear un nuevo proyecto Java Web

```
Archivo -> Proyecto Nuevo -> Java Web -> Web Application [Siguiente]

        Project Name: sumador
        Use dedicated folder for storing libraries: ./lib

        Add to Enterprise Application: <none>
        Server: GlassFish 3.1.2
        Java EE version: Java EE 6 web
        Context path: sumador

        Seleccionar en Frameworks "Java Server Faces"
        Server Library: JSF 2.1
        Configuration -> Prefered Page Language: Facelets
        Components -> Components Suite: Primefaces
```

2. Uso básico de JSF

2.1. Calculadora básica

2.1.1. Componentes a utilizar

h:head, h:body Replican las etiquetas `<head>` y `<body>` de HTML. Utilizados para cierto tipo de configuraciones/inicializaciones (recursos CSS, JavaScript, etc)

h:form Delimita un formulario JSF (todos los componentes JSF susceptibles de generar acciones y/o eventos deben incuirse dentro de estos componentes)

h:panelGrid Contenedor de componentes, usado para controlar su posición en el formulario

h:outputLabe Componente que genera un `<label>` de HTML

h:inputText Componente de captura de datos (caja de texto HTML) (vinculado a atributos de los `@ManagedBean` mediante el atributo `value="..."`)

h:commandButton Botón sobre el que usuario podrá actuar

- el comportamiento por defecto (cuando no se usa AJAX) es que genere una petición HTTP POST que desencade en el servidor el ciclo de procesamiento de JSF

- atributo `actionListener="..."` especifica un método de un `|ManagedBean` que actuará como manejador del evento de pulsación del botón
- atributo `action="..."` especifica la siguiente vista JSF a la que se navegará, pude ser un String o un método "de acción" de un `@ManagedBean` que devuelva un String con una indicación de navegación.

2.1.2. Managed Beans

1. Crear un nuevo paquete de nombre `controladores`

```
Sobre el proyecto 'sumador' -> Source packages
    [Botón derecho] -> Nuevo -> Java Package
    Crear el paquete 'controladores'
```

2. Crear una clase *SumadorController* en `controladores`

```
Sobre el proyecto 'sumador' -> Source packages -> controladores
    [Botón derecho] -> Nuevo -> Java Class
    Crear la clase 'SumadorController'
```

SumadorController.java

```
@ManagedBean(name = "sumadorController")
@SessionScoped
public class SumadorController implements Serializable {
    private double operador1;
    private double operador2;
    private double resultado;

    ...
    // constructor      (automático con Fuente -> Insertar código -> Constructor)
    // getter y setter  (automático con Fuente -> Insertar código -> Getter y Setter)
    ...

    public void doSuma() {
        sumar();
    }

    private void sumar() {
        resultado = operador1 + operador2;
    }
}
```

Anotaciones:

@ManagedBean Marca la clase como `@ManagedBean`, opcionalmente pude asociársele un nombre a través del cual se vinculará con los componentes de las vistas JSF

Indica que las instancias de la clase serán creadas y gestionadas por el framework JSF

Nota: si se usa el API CDI (*Context and Dependency Injection*) se puede utilizar la anotación `@Named` que proporciona la misma finalidad junto con otras características añadidas

@SessionScoped Especifica el alcance ("tiempo de vida") del objeto, en este caso indica que es un objeto "de sesión" (otras alternativas: `@RequestScoped`, `@ApplicationScoped`, `@ViewScoped`)

Importante: si se generan los `import` automáticamente, asegurad que se utilicen los de JSF (`javax.faces.bean.[...]`) y no los de CDI (`javax.enterprise.context.[...]`)

2.1.3. Definición de la vista JSF

index.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dt
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:p="http://primefaces.org/ui">
    <h:head>
        <title>ESEI Dojo</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="2">
                <h:outputLabel value="Operador 1:" for="op1"/>
                <h:inputText id="op1" value="#{sumadorController.operador1}" label="Operador 1"/>

                <h:outputLabel value="Operador 2:" for="op2"/>
                <h:inputText id="op2" value="#{sumadorController.operador2}" label="Operador 2"/>

                <h:outputLabel value="Resultado:" for="res"/>
                <h:inputText id="res" value="#{sumadorController.resultado}" readonly="true"/>

                <h:commandButton value="sumar"
                                 actionListener="#{sumadorController.doSuma()}"
                                 action="index.xhtml"/>
            </h:panelGrid>
        </h:form>
    </h:body>
</html>
```

2.2. Mejora 1: incluir conversores

2.2.1. Componentes a utilizar

h:messages Especifica un "espacio" donde se mostarrán los mensajes de error generados durante el procesamiento en el servidor de la petición enviada por el navegador cliente (errores de validación, de formato/conversión, generados por la propia aplicación, etc)

f:convertNumber Indica que los datos vinculados a un componente de entrada o de salida (`<h:outputText>`, `<h:inputText>`, etc) al que se le adosa esta etiqueta serán tratados/formteados como un número con las restricciones que se fijen.

2.2.2. Managed Beans

```
< nada que modificar >
```

2.2.3. Definición de la vista JSF

index.xhtml

```
...
        <h:form>
            <h:messages style="color:red"/>

            <h:panelGrid columns="2">
                <h:outputLabel value="Operador 1:" for="op1"/>
```

```
            <h:inputText id="op1" value="#{sumadorController.operador1}" label="Operador 1" required="true">
                <f:convertNumber />
            </h:inputText>

            <h:outputLabel value="Operador 2:" for="op2"/>
            <h:inputText id="op2" value="#{sumadorController.operador2}" label="Operador 2" required="true">
                <f:convertNumber />
            </h:inputText>

            <h:outputLabel value="Resultado:" for="res"/>
            <h:inputText id="res" value="#{sumadorController.resultado}" readonly="true">
                <f:convertNumber />
            </h:inputText>

            <h:commandButton value="sumar"
                             actionListener="#{sumadorController.doSuma()}"
                             action="index.xhtml"/>

        </h:panelGrid>
    </h:form>
...
```

2.3. Mejora 2: incluir histórico de operaciones

2.3.1. Componentes a utilizar

h:dataTable Especifica un componente de tipo tabla que se alimentará de un array o `Collection` que resida en un atributo de un `@ManagedBean`

- atributo `value="..."`: especifica el atributo que contiene la lista de elementos a partir cuales se generarán las filas de la tabla
- atributo `var="..."`: declara una pseudovariable que referenciará al elemento de la lista vinculado a la fila actual

h:column Especifica los contenido de cada columna de la tabla. Es un contenedor de componentes JSF que podrán referenciar a la pseudovariable declarada en el atributo `var`.

f:facet Especifica parámetros o características de otros componenetes JSF. En este caso se usa para especificar la cabecera de cada columna.

2.3.2. Managed Beans

1. Crear una nueva clase *Operación* en el paquete `controladores`

```
public class Operacion {
    private int id;
    private double operando1;
    private double operando2;
    private double resultado;


    // constructor      (automático con Fuente -> Insertar código -> Constructor)
    public Operacion(int id, double operando1, double operando2, double resultado) {
        this.id = id;
        this.operando1 = operando1;
        this.operando2 = operando2;
        this.resultado = resultado;
    }

    // getter y setter  (automático con Fuente -> Insertar código -> Getter y Setter)
```

```
    ...
}
```

2. Añadir un atributo `operaciones` con la lista de operaciones realizadas y un contador `contadorOperaciones` en la clase *SumadorController* y un método `doEliminarOperacion()`

```
@ManagedBean(name = "sumadorController")
@SessionScoped
public class SumadorController implements Serializable {
    private double operador1;
    private double operador2;
    private double resultado;

    private List<Operacion> operaciones = new ArrayList<Operacion>();
    private int contador = 1

    // getter y setter para 'operaciones'  (automático con Fuente -> Insertar código -> Getter y Setter)
    ...

    private void sumar() {
        resultado = operador1 + operador2;

        // anadir operacion a la lista + incr. contador
        operaciones.add(new Operacion(contador, operador1, operador2, resultado));
        contador++;
    }

    public void doEliminar(Operacion operacion){
        operaciones.remove(operacion);
    }
}
```

2.3.3. Definición de la vista JSF

index.xhtml : añadir después de `</h:panelGrid>`

```
...
           <h:dataTable value="#{sumadorController.operaciones}" var="operacion" >
               <f:facet name="header">
                   Historico de sumas ....
               </f:facet>

               <h:column>
                   <f:facet name="header"> ID </f:facet>
                   <h:outputText value="#{operacion.id}"/>
               </h:column>

               <h:column>
                   <f:facet name="header"> Operando 1 </f:facet>
                   <h:outputText value="#{operacion.operando1}"/>
               </h:column>

               <h:column>
                   <f:facet name="header"> Operando 2 </f:facet>
                   <h:outputText value="#{operacion.operando2}"/>
               </h:column>

               <h:column>
                   <f:facet name="header"> Resultado </f:facet>
                   <h:outputText value="#{operacion.resultado}"/>
               </h:column>
```

```
                <h:column>
                    <h:commandButton value="eliminar"
                                     actionListener="#{sumadorController.doEliminar(operacion)}"
                                     action="index.xhtml"/>
                </h:column>

            </h:dataTable>
        </h:form>
...
```

3. Uso de Primefaces

3.1. Versión AJAX del "sumador"

3.1.1. Componentes a utilizar

p:panel, p:outputPanel, p:panelGrid Paneles y componentes Primefaces que actúan como contenedores de componentes con los estilos y "skins" de Primefaces

p:commandButton Variante de `<h:commandLink>` con soporte para interaciones AJAX.

Permite especificar que porciones de la vista JSF deben de ser enviadas al servidor para procesar sus modificaciones (atributo `process`) e indicar que elementos de la vista deberán ser actualizados una vez completada la acción (atributo `update`)

p:inputText Variante de `<h:inputText>` que se integra con los estilos y "skins" de Primefaces

p:messages Variante mejorada de `<h:messages>`

p:datagrid, p:column Dispone un conjunto de elementos de un `@ManagedBean` (arrays ó *Collection*) en una parrilla. Cada celda de la parrilla contendrá uno o más componentes Primefaces. Permite paginación.

3.1.2. Definición de la vista JSF

1. Previo: organizar la vista en paneles

```
<h:form>

   <p:messages id="errores"  autoUpdate="true"/>

   <p:panel id="panelSumas" header="Panel de sumas" >
      ...
   </p:panel>

   <p:panelGrid columns="2">
      <p:panel id="panelHistorial" header="Historial de sumas">
         ...
      </p:panel>

      <p:panel id="panelFavoritas" header="Sumas favoritas">
         ...
      </p:panel>
   </p:panelGrid>

</h:form>
```

2. Componentes del "panel de sumas"

```
...
<p:panel id="panelSumas" header="Panel de sumas" >
   <p:panelGrid  columns="2" >
      <h:outputLabel value="Operador 1:" for="op1"/>
      <p:inputText id="op1" value="#{sumadorController.operador1}" required="true" label="Operador 1">
         <f:convertNumber />
      </p:inputText>

      <h:outputLabel value="Operador 2:" for="op2"/>
      <p:inputText id="op2" value="#{sumadorController.operador2}" required="true" label="Operador 2">
         <f:convertNumber />
      </p:inputText>

      <h:outputLabel value="Resultado:" for="res"/>
      <p:inputText id="res" value="#{sumadorController.resultado}" readonly="true">
          <f:convertNumber />
      </p:inputText>

      <p:commandButton value="sumar" ajax="true"
                 actionListener="#{sumadorController.doSuma()}"
                 process="panelSumas"
                 update="panelSumas panelHistorial errores"/>

    </p:panelGrid>
</p:panel>
...
```

3. Componentes del "historial de sumas"

```
   ...
   <p:panelGrid columns="2">
      <p:panel id="panelHistorial" header="Historial de sumas">
         <p:dataGrid id="datagridDatos"
                     columns="3" paginator="true" rows="6"
                     emptyMessage="Sin operaciones"
                     value="#{sumadorController.operaciones}"
                     var="operacion">
            <p:column>
               <p:panel id="panel" header="#{operacion.id}">
                  <h:panelGrid columns="2">
                     <h:outputLabel value="Op1:"/>
                     <h:outputText value="#{operacion.operando1}"/>

                     <h:outputLabel value="Op2:"/>
                     <h:outputText value="#{operacion.operando2}"/>

                     <h:outputLabel value="Res:"/>
                     <h:outputText value="#{operacion.resultado}"/>
                  </h:panelGrid>
               </p:panel>

               <p:draggable for="panel" helper="clone" revert="true" />
            </p:column>
         </p:dataGrid>
      </p:panel>
      ...


   </p:panelGrid>
   ...
```

3.2. Añadir Drag-and-Drop

3.2.1. Componentes a utilizar

p:dataTable, p:column Componentes Primefaces para la definición de tablas, soportan paginación de resultados, ordenación, selección múltiple, etc

p:draggable Se vincula a un componente (normalmente un contenedor) indicando que es "arrastrable". El modo de selección y el comportamiento de los "arrastrables" es configurable.

p:droppable Se vincula a un componente (normalmente un contenedor) que puede ser el destino donde depositar los "arrastrables".

Si se desea vincular el elemento "arrastrado" con los datos concretos de algún componente de tipo lista (`<p:dataTable>`, `<p:datagrid>`) se puede usar el atributo `datasource`.

p:ajax Configura los eventos AJAX generados por el componente padre (en este caso un evento *DropEvent* de un `<p:droppable>`).

El atributo `listener` indica el método manejador del evento en un `@ManagedBean` y el atributo `update` especifica los componentes de la vista a actualizar una vez completada su invocación.

3.2.2. Managed Beans

1. Incluir un atributo `operacionesSeleccionadas` en `SumadorController` con sus respectivos `get()` y `set()`.

   Contendrá la lista de "sumas favoritas".

2. Añadir un manejador de eventos para el evento Drop (recupera el objeto *Operación* seleccionado y lo añade a la lista `operacionesSeleccionadas`)

```
@ManagedBean(name = "sumadorController")
@SessionScoped
public class SumadorController implements Serializable {
    ...

    private List<Operacion> operacionesSeleccionadas = new ArrayList<Operacion>();
    // getter y setter para 'operacionesSeleccionadas'  (automático con Fuente -> Insertar código -> Getter y Setter)
    ...

    public void doDropOperacion(DragDropEvent evento){
        Operacion seleccionada = (Operacion) evento.getData();
        operacionesSeleccionadas.add(seleccionada);
    }
}
```

3.2.3. Definición de la vista JSF

1. Vincular un componente `<p:draggable>` al panel incluido dentro de cada celda del `<p:dataGrid>`
2. Crear un panel que incluya un `<p:dataTable>` vinculado al atributo `operacionesSeleccionadas`
3. Vincular un componente `<p:droppable>` al panel de "sumas favoritas", enlazarlo con el `<p:dataGrid>` con la lista completa de operaciones y configurarlo para que su evento *onDrop* invoque mediante AJAX el manejador de eventos (atributo `listener` de `<p:ajax>`)

```
...
 <p:panelGrid columns="2">
   <p:panel id="panelHistorial" header="Historial de sumas">
      <p:dataGrid  ....>
         <p:column>
            <p:panel id="panel" header="#{operacion.id}">
```

```
            ...
         </p:panel>

         <p:draggable for="panel" helper="clone" revert="true" />

      </p:column>
   </p:dataGrid>
</p:panel>

<p:panel id="panelFavoritos" header="Sumas favoritas">
   <p:outputPanel id="dropArea">
      <p:dataTable value="#{sumadorController.operacionesSeleccionadas}"
                   var="operacionSeleccionada"
                   paginator="true" rows="5"
                   emptyMessage="Sin operaciones favoritas">
         <p:column>
            <f:facet name="header">ID</f:facet>
            <h:outputText value="#{operacionSeleccionada.id}"/>
         </p:column>

         <p:column>
            <f:facet name="header">op1</f:facet>
            <h:outputText value="#{operacionSeleccionada.operando1}"/>
         </p:column>

         <p:column>
            <f:facet name="header">op2</f:facet>
            <h:outputText value="#{operacionSeleccionada.operando2}"/>
         </p:column>

         <p:column>
            <f:facet name="header">res</f:facet>
            <h:outputText value="#{operacionSeleccionada.resultado}"/>
         </p:column>
      </p:dataTable>
   </p:outputPanel>

   <p:droppable for="panelFavoritos" datasource="datagridDatos" tolerance="touch" >
      <p:ajax listener="#{sumadorController.onDropOperacion}" update="panelFavoritos" />
   </p:droppable>
</p:panel>
</p:panelGrid>
...
```