



Componentes Web

Sesión 8 - *Facelets*, JSTL y lenguaje de expresiones



Índice

- JSF y Facelets
- Lenguaje de expresiones
- Librería core de JSTL
- Librería HTML de JSF
- Librería UI de Facelets



JSF

- *Framework* para la creación de aplicaciones web. Proporciona:
 - API con componentes de la interfaz (campos de texto, botones, listas, etc) en el lado del servidor
 - Librerías de etiquetas para añadir los componentes a una página web
- Los componentes son elementos configurables y reutilizables
 - Todos heredan de `UIComponentBase`
 - Implementa el comportamiento pero no la forma de mostrarlo
 - Podremos aplicar diferentes *renderers* a cada componente para cambiar la forma de mostrarlo
- Las etiquetas de JSF relacionan componentes con *renderers*
 - `<h:selectOneListbox>`, `<h:selectOneMenu>` y `<h:selectOneRadio>` son un mismo tipo de componente (`UISelectOne`) con diferentes *renderers*



Facelets

- **Páginas XHTML** en las que podemos utilizar:
 - **Librerías de etiquetas** de Facelets, JSF y JSTL: Componentes de la interfaz y forma de validarlos
 - **Lenguaje de expresiones (EL)**: Relaciona componentes con datos (*managed beans*)
 - **Plantillas** de componentes y páginas: Definen la estructura del contenido
- Para utilizar Facelets deberemos
 - Mapear el servlet de procesamiento de Facelets a un patrón de URL (`*.xhtml`)
 - Crear *managed beans* para acceder a los datos de la aplicación a través de ellos
 - Crear páginas utilizando las librerías de componentes

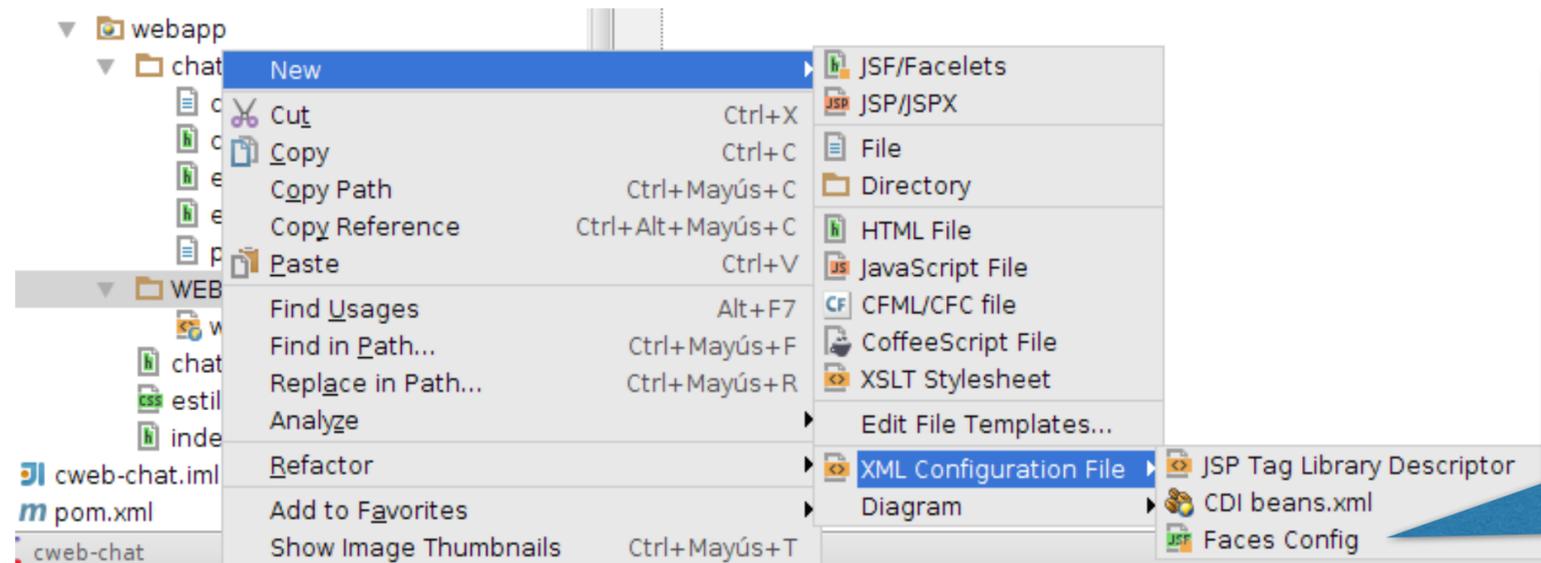


Configuración del servlet de *Facelets*

- Configuramos y mapeamos `FacesServlet` en el descriptor de despliegue

```
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.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>
```

- Añadimos el fichero de configuración de JSF (`faces-config.xml`) a `WEB-INF`

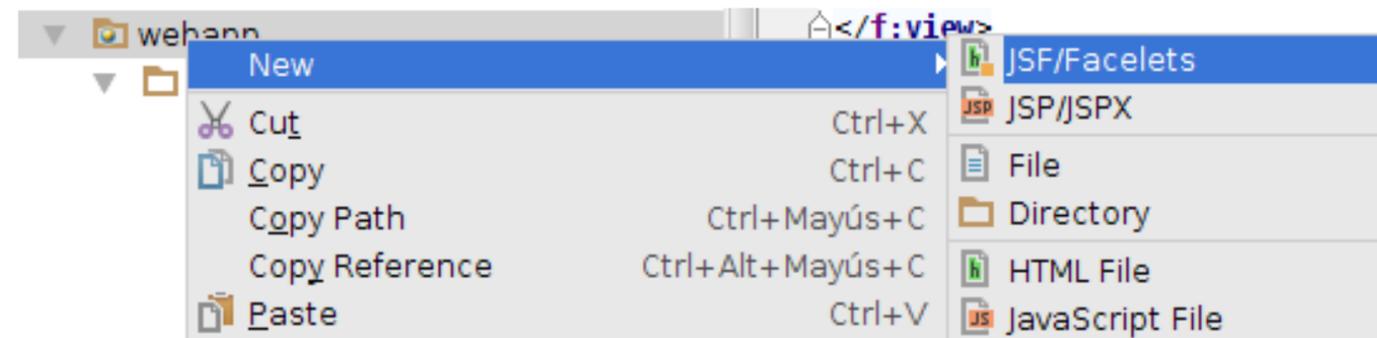


Al añadir *Faces Config* con IntelliJ (*New > XML Configuration File > Faces Config*) automáticamente se añade el mapeo de *Faces Servlet* al descriptor de despliegue



Creación de un *Facelet*

- Podemos crear *Facelets* desde IntelliJ con *New > JSF/Facelets*



- La estructura básica es la siguiente:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html lang="en"
    xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html">

    <h:head>
        <title>Mi primer Facelet</title>
    </h:head>
    <h:body>
        <!-- Contenido de la página -->
    </h:body>
</html>
```

Declaración de librerías de etiquetas



Declaración de librerías de etiquetas

Librería	URI	Prefijo	Descripción
JSF HTML Tag Library	http://xmlns.jcp.org/jsf/html	h	Componentes de la UI de la página
JSF Core Tag Library	http://xmlns.jcp.org/jsf/core	f	Funciones y acciones
JSTL Core Tag Library	http://xmlns.jcp.org/jsp/jstl/core	c	Etiquetas de propósito general de JSTL
JSTL Functions Tag	http://xmlns.jcp.org/jsp/jstl/functions	fn	Funciones de JSTL

- Podemos declararlas con prefijos distintos del prefijo por defecto

```
<html lang="en"
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://xmlns.jcp.org/jsf/html"
  xmlns:c="http://xmlns.jcp.org/jsp/jstl/core">
```

`<h:body>`

```
<html lang="en"
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:html="http://xmlns.jcp.org/jsf/html"
  xmlns:core="http://xmlns.jcp.org/jsp/jstl/core">
```

`<html:body>`



Introducción al lenguaje de expresiones (EL)

- Desde JSP 2.0 se dispone de un lenguaje de expresiones que permite realizar varias tareas, algunas de las cuales se hacían hasta el momento con scriptlets `<%= . . . %>`
- Antes, este lenguaje sólo estaba disponible en JSTL, una librería de etiquetas adicionales para JSP

```
<p>La variable edad se ha establecido a ${edad}</p>  
<h4>El parametro nombre vale ${param.nombre}</h4>  
<c:if test="`${persona.edad} > 18`">  
    ...  
</c:if>
```



Atributos y expresiones

- El lenguaje se puede utilizar en atributos de etiquetas, o dentro de la parte HTML de la página, y se invoca con el elemento `${ ... }`

```
${expresion}
```

- Podemos utilizar estos elementos:
 - Solos en atributos de etiquetas

```
<pref:etiq value="${expresion}"/>
```

- Combinados con otros elementos en atributos JSTL

```
<pref:etiq value="texto${expr1} y ${expr2} texto"/>
```

- Dentro del contenido HTML de la página

```
<h4>Hola, esto es la variable ${miVar}</h4>
```



Operadores

- Dentro de las expresiones podemos utilizar operadores:

- *Operadores* [] y . son equivalentes:

```
${expr[campo]}=${expr.campo}
```

- *Operadores aritméticos*: +, -, *, /, div, mod, %
- *Operadores relacionales*: >, gt, <, lt, >=, ge, <=, le, ==, eq, !=, ne
- *Operadores lógicos*: &&, and, ||, or, !, not
- *Operador* “empty”: para indicar si un elemento es nulo o vacío (devuelve true) o no (devuelve false)

```
<c:if test="${empty A}">  
  ...  
</c:if>
```



Nombres de variables

- Los nombres de variables que pongamos se evalúan según `pageContext.findAttribute(...)`, buscando la variable en la página, petición, sesión, etc. Si no lo encuentra, devuelve `null`
- Hay algunas palabras reservadas que no podemos usar como nombres: `and`, `eq`, `or`, `lt...` etc
- Cuando como nombre de la variable utilizamos ciertos nombres implícitos, se devuelve el objeto al que pertenecen:

```
${param.id}
```

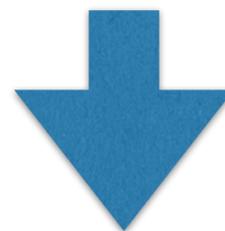
- Devolvería el valor del parámetro `id` de la petición
- Otros nombres implícitos: `header`, `sessionScope`, `cookie...`



Managed beans y lenguaje de expresiones

- Para poder acceder a los *managed beans* desde EL debemos darles un nombre
- Esto lo haremos mediante la anotación @Named
 - Por defecto será el nombre de la clase cambiando la primera letra por minúscula

```
@Named("hola")
public class HolaMundo {
    public String getSaludo() {
        return "Hola mundo!";
    }
}
```



```
${hola.saludo}
```



Ejemplos

```
// Devuelve la suma de las dos variables  
${num1 + num2}  
  
// true si v1 es distinto a v2 y v3 menor que v4  
${v1 ne v2 and v3 < v4}  
  
// obtiene el atributo profile de la sesión  
${sessionScope.profile}  
  
// obtiene el parámetro password de la petición  
${param.password}  
  
// true si el parámetro nombre de la petición está vacío o nulo  
${empty param.nombre}
```



Librerías de tags y JSTL

- Una librería de etiquetas (*taglib*) es un conjunto de etiquetas HTML personalizadas que permiten encapsular acciones mediante código Java
- JSTL (*Java Server Pages Standard Tag Library*)
 - Grupo de librerías de etiquetas estándar que encapsula varias funcionalidades: SQL, XML, internacionalización...
 - En Facelets están disponibles la librería Core y la librería de Funciones
 - Además, dispone de un lenguaje de expresiones que utiliza en sus etiquetas (el mismo que tiene JSP 2.0)
 - Al ser estándar funciona igual en todos los servidores, y los contenedores pueden reconocerla y optimizar sus implementaciones
 - Disponible a partir de servlets 2.3 y JSP 1.2



Librería Core de JSTL

- La librería *Core* incluye tags de propósito general para:
 - Evaluación de expresiones
 - Establecimiento/obtención de valores de parámetros
 - Sentencias de control de flujo: condiciones, iteradores...
 - Funciones de acceso a URLs
- Normalmente, los tags de esta librería se utilizan con el prefijo "c"



El tag *out*

- Se utiliza para evaluar el resultado de una expresión y colocarlo en la salida JSP

```
<c:out value="valor" [escapeXML="true|false"] [default="valor"]/>  
<c:out value="valor" [escapeXML="true|false"]>  
    Valor por defecto  
</c:out>
```

- Ejemplo:

```
<c:out value="${datos.ciudad}" default="desconocida"/>
```



El tag `set`

- Establece el valor de un atributo en cualquier sección (`param`, `header`, `cookie`...)

```
<c:set value="valor" var="variable" [scope="page|request|session|application"]/>
<c:set var="variable" [scope="page|request|session|application"]>
  Valor
</c:set>

<c:set value="valor" target="objeto" property="propiedad"/>
<c:set target="objeto" property="propiedad">
  Valor
</c:set>
```

- Ejemplo:

```
<c:set var="foo" value="2"/>
<c:set value="19" target="{persona}" property="edad"/>
<c:out value="foo vale {foo} y edad vale {persona.edad}"/>
```



El tag *if*

- Ejecuta su código si se cumple una condición, o guarda el valor de la comparación en una variable

```
<c:if test="condicion" var="variable" [scope="page|request|session|application"]/>  
  
<c:if test="condicion" [var="variable"  
    [scope="page|request|session|application"]>  
    Cuerpo  
</c:if>
```

- Ejemplo:

```
<c:if test="${visitas > 1000}">  
<h1>¡Mas de 1000 visitas!</h1>  
</c:if>
```



El tag *choose*

- Ejecuta una de las opciones `when` que tiene (la que cumpla la condición), o la `otherwise` si ninguna la cumple. Similar al *switch* de C o Java

```
<c:choose>
  <c:when test="condicion1">codigo1</c:when>
  <c:when test="condicion2">codigo2</c:when>
  ...
  <c:when test="condicionN">codigoN</c:when>
  <c:otherwise>codigo</c:otherwise>
</c:choose>
```

- Ejemplo:

```
<c:choose>
  <c:when test="{a < 0}"><h1>a menor que 0</h1></c:when>
  <c:when test="{a > 10}"><h1>a mayor que 10</h1></c:when>
  <c:otherwise><h1>a entre 1 y 10</h1></c:otherwise>
</c:choose>
```



El tag *forEach*

- Repite su código recorriendo un conjunto de objetos o un número de iteraciones

```
<c:forEach [var="variable" items="conjunto" [varStatus="variableEstado"]  
           [begin="comienzo" [end="final" [step="incremento"]]>  
    codigo  
</c:forEach>  
  
<c:forEach [var="variable" [varStatus="variableEstado"] begin="comienzo"  
           end="final" [step="incremento"]>  
    codigo  
</c:forEach>
```

- Ejemplo:

```
<c:forEach var="item" items="{cart.items}">  
    <c:out value="{item.valor}"/>  
</c:forEach>
```



El tag *forTokens*

- Similar a `forEach` pero recorre una cadena, separando por los delimitadores indicados
- La sintaxis es la misma que `forEach`, pero con un atributo `delim` que es obligatorio, e indica los delimitadores de la cadena
- Ejemplo:

```
<c:forTokens var="item" items="un#token otro#otromas" delims="# ">  
    <c:out value="{item}"/>  
</c:forTokens>
```

- Sacaría 4 tokens: "un", "token", "otro" y "otromas"



Los tags *import* y *param*

- `import` se utiliza para importar el contenido de una URL
- Internamente, puede utilizar tags `param` (otros tags también pueden utilizarlo) para especificar parámetros de la URL

- Ejemplo:

```
<c:import url="http://localhost/mipagina.jsp" var="varurl">  
  <c:param name="id" value="12"/>  
</c:import>  
  
<c:out value="{varurl}"/>
```

- Equivaldría a importar el contenido de <http://unapagina.com?id=12>



Librería de Funciones

- La librería de *funciones* (`fn`) recopila una serie de funciones que introducir dentro del lenguaje de expresiones (sustituir en cadenas, concatenar, etc).

```
La cadena tiene <c:out value="{fn:length(miCadena)}"/> caracteres
```



Librería HTML de JSF

- Librería de componentes de la interfaz
 - Se renderizan como HTML
 - Se pueden vincular con *managed beans* mediante EL

Se vincula el valor del campo de texto con la propiedad `precio` del *managed bean* `articulo`, y se valida que sea una cantidad positiva

```
<h:inputText id="precio"
             size="4"
             value="#{articulo.precio}"
             title="Cantidad">
    <f:validateLongRange minimum="0"/>
</h:inputText>
```

- Atributos comunes

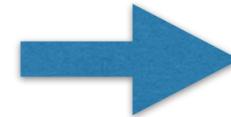
Atributo	Función
<code>id</code>	Identifica al componente de forma única.
<code>rendered</code>	Podemos especificar una condición mediante EL que indica si el componente se debe mostrar o no.
<code>style</code>	Especifica el estilo CSS.
<code>styleClass</code>	Especifica una clase de estilo.



Recursos

- Artefactos necesarios para que la página se muestre correctamente
 - Hojas de estilo, imágenes, *scripts*, etc
- Las ubicaciones estándar son
 - Directorio `resources` en la raíz del contexto
 - Directorio `META-INF/resources` dentro del *classpath*
- Podemos incluir una hoja de estilo como recurso

```
<h:outputStylesheet library="css" name="estilo.css"/>
```



```
/resources/css/estilo.css
```

- También podemos especificar la ubicación mediante lenguaje de expresiones

```
<h:graphicImage value="#{resource['imagenes:logo.gif']}/>
```



```
/resources/imagenes/logo.gif
```



Plantillas

Especificamos mediante `ui:insert` las secciones de la página

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    xmlns:h="http://xmlns.jcp.org/jsf/html">

    <h:head>
        <meta http-equiv="Content-Type"
            content="text/html; charset=UTF-8" />
        <h:outputStylesheet library="css" name="estilo.css"/>
        <title>Plantilla con Facelets</title>
    </h:head>

    <h:body>
        <div id="top" class="top">
            <ui:insert name="cabecera">Cabecera</ui:insert>
        </div>
        <div>
            <div id="left">
                <ui:insert name="menu">Menú lateral</ui:insert>
            </div>
            <div id="content" class="content">
                <ui:insert name="cuerpo">Cuerpo de la página</ui:insert>
            </div>
        </div>
    </h:body>
</html>
```

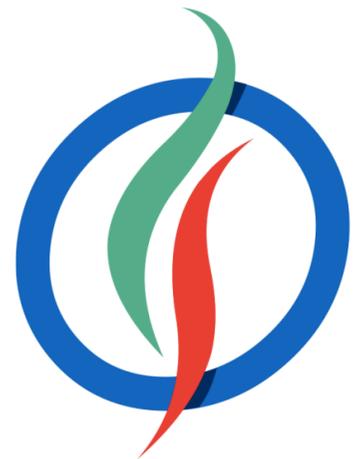


Aplicación de plantillas

Con `ui:composition` aplicamos una plantilla al documento

Con `ui:define` damos contenido a cada sección de la plantilla

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:body>
    <ui:composition template="./plantilla.xhtml">
      <ui:define name="cabecera">
        Título de la página
      </ui:define>
      <ui:define name="menu">
        <h:outputLabel value="Menú"/>
      </ui:define>
      <ui:define name="cuerpo">
        <h:graphicImage value="#{resource['images:logo.gif']}/>
        <h:outputText value="2014 (c) DCCIA"/>
      </ui:define>
    </ui:composition>
  </h:body>
</html>
```



¿Preguntas?