JSF 2

Víctor Herrero Cazurro

Contenidos

1. Introducción
2. Ciclo de vida de una petición
3. Vistas
4. Managed beans
5. Eventos y Acciones
6. Expresiones EL
7. Reglas de navegación
8. Internacionalización – i18n
9. Javascript y Ajax
10. Conversores
11. Validadores
12. Facelets
13. Componentes personalizados

¿Qué es un framework?

- Herramienta diseñada para facilitar el desarrollo de software.
- Nos ayuda a cumplir los estándares.
- Provee de una estructura y de una metodología de trabajo.

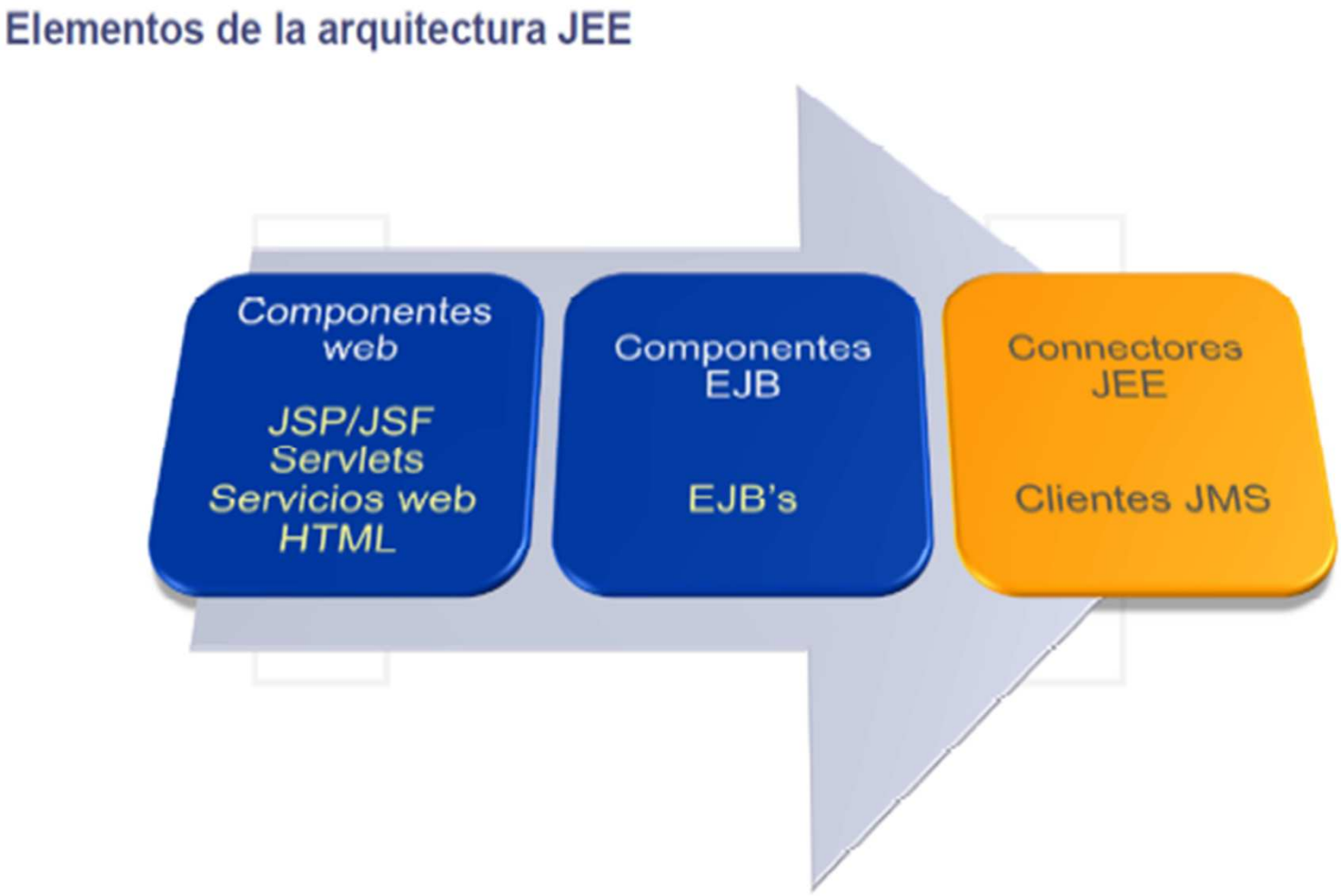
¿Por qué JSF?

- Muy sencillo de utilizar. Se pueden crear interfaces de usuario con mucha facilidad.
- Ofrece una clara separación entre la presentación y el comportamiento de una pagina.
- Facilidad para manejo de componentes, proceso de datos, validaciones y eventos.

¿Por qué JSF?

- Mecanismo de eventos robusto
- Soporte de “Render Kit” para distintos clientes. El que se incluye en la implementación de referencia es el de HTML.
- Componentes modulares fácilmente sustituibles.

Arquitectura JEE



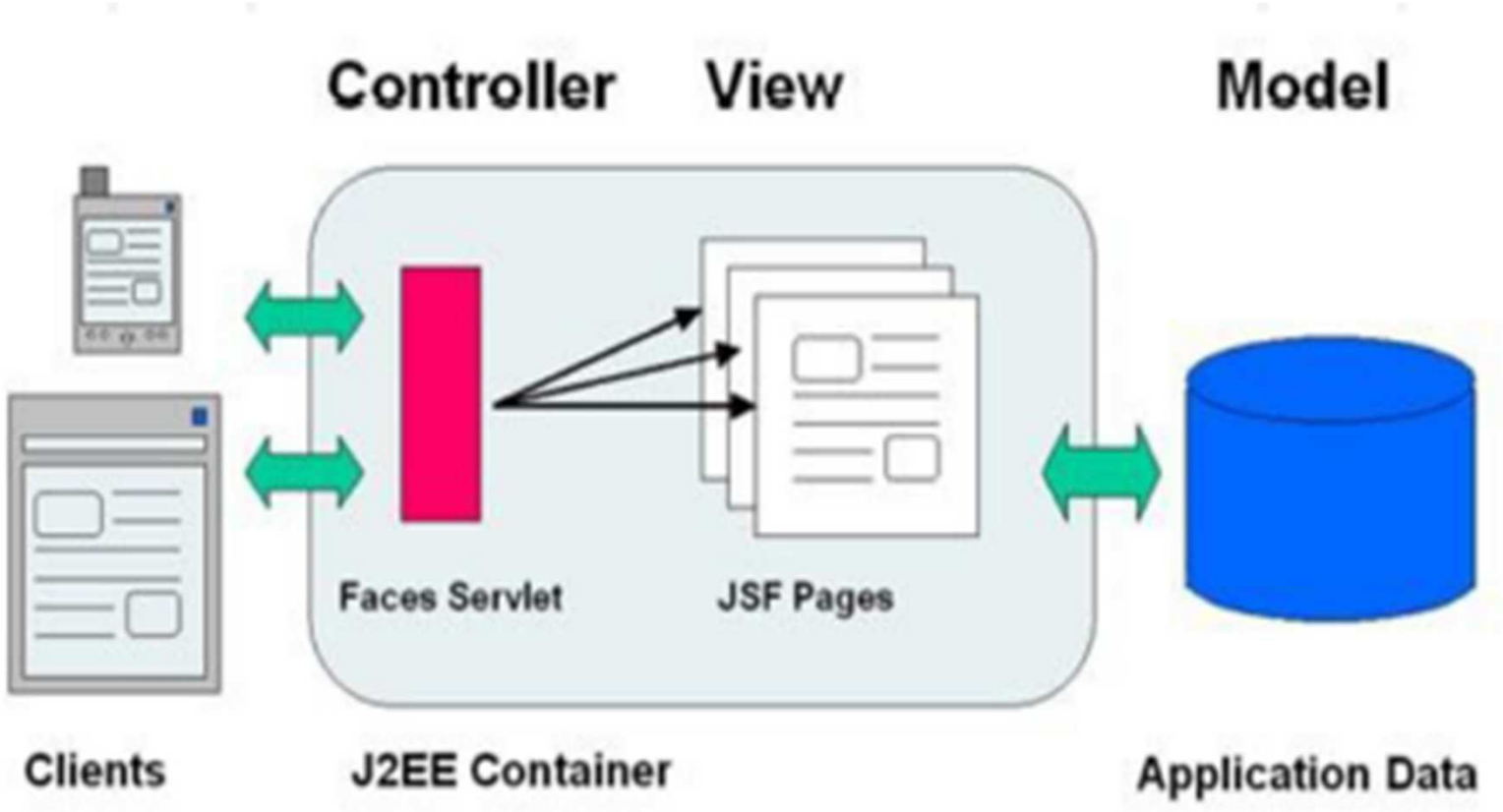
Arquitectura JEE

- Java Specification Reference (JSR)
- El grupo “jcp.org” es el encargado de dirigir/mantener las especificaciones de las API’s de Java

Arquitectura JEE

- **JSR-127** define y libera la especificación de JavaServer Faces 1.1 en 2004 en el que participó Craig McClanahan que lideró Apache Tomcat y Struts entre otros.
- **JSR-252** define y libera la especificación JSF 1.2 en 2006 que corresponde a los Servlets 2.5/JSP 2.1
- **JSR-314** comienza en mayo de 2007 y planifica y libera la especificación JSF 2.0 en 2009 con soporte para Ajax.

Arquitectura MVC



Preparación del entorno

- JSF cubre la capa web de los proyectos, por lo que se necesita un proyecto tipo Web.
- El proyecto deberá tener en el **classpath** las librerías de la implementación a emplear de JSF.
  - Si es un servidor JEE, el servidor podrá proveer dichas dependencias.
  - Sino el proyecto deberá incluirlas.

Preparación del entorno

- Las librerías necesarias dependerán de la implementación.
- Existen dos implementaciones de jsf
  - Mojarra (Implementación de referencia de Oracle).
  - Apache MyFaces.

Preparación del entorno

- Para añadir **Mojarra** al proyecto con Maven

```
<dependency>
    <groupId>com.sun.faces</groupId>
    <artifactId>jsf-api</artifactId>
    <version>2.2.10</version>
</dependency>

<dependency>
    <groupId>com.sun.faces</groupId>
    <artifactId>jsf-impl</artifactId>
    <version>2.2.10</version>
</dependency>
```

Preparación del entorno

- Para añadir **Apache Myfaces** al proyecto con Maven

```
<dependency>
    <groupId>org.apache.myfaces.core</groupId>
    <artifactId>myfaces-api</artifactId>
    <version>2.2.7</version>
</dependency>
<dependency>
    <groupId>org.apache.myfaces.core</groupId>
    <artifactId>myfaces-impl</artifactId>
    <version>2.2.7</version>
</dependency>
```

Preparación del entorno

- Y también existen múltiples extensiones de JSF, como
  - Oracle ADF
  - RichFaces
  - IceFaces
  - PrimeFaces

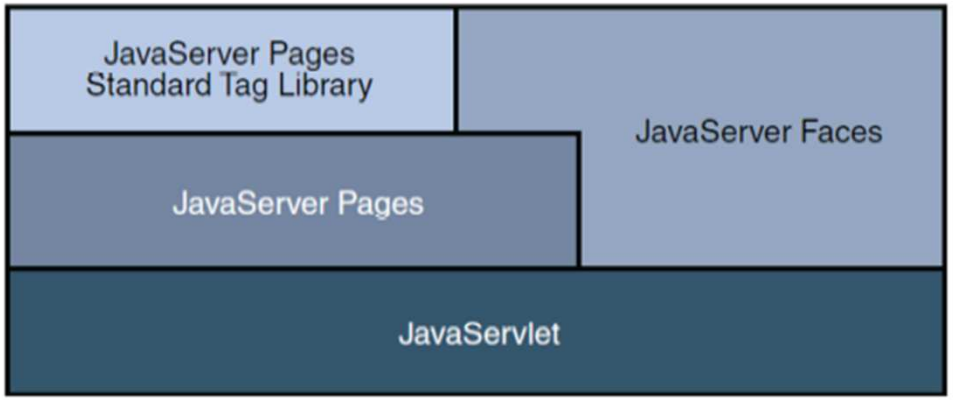
Preparación del entorno

- Para añadir **PrimeFaces** al proyecto con Maven

```
<dependency>
    <groupId>org.primefaces</groupId>
    <artifactId>primefaces</artifactId>
    <version>5.1</version>
</dependency>
```

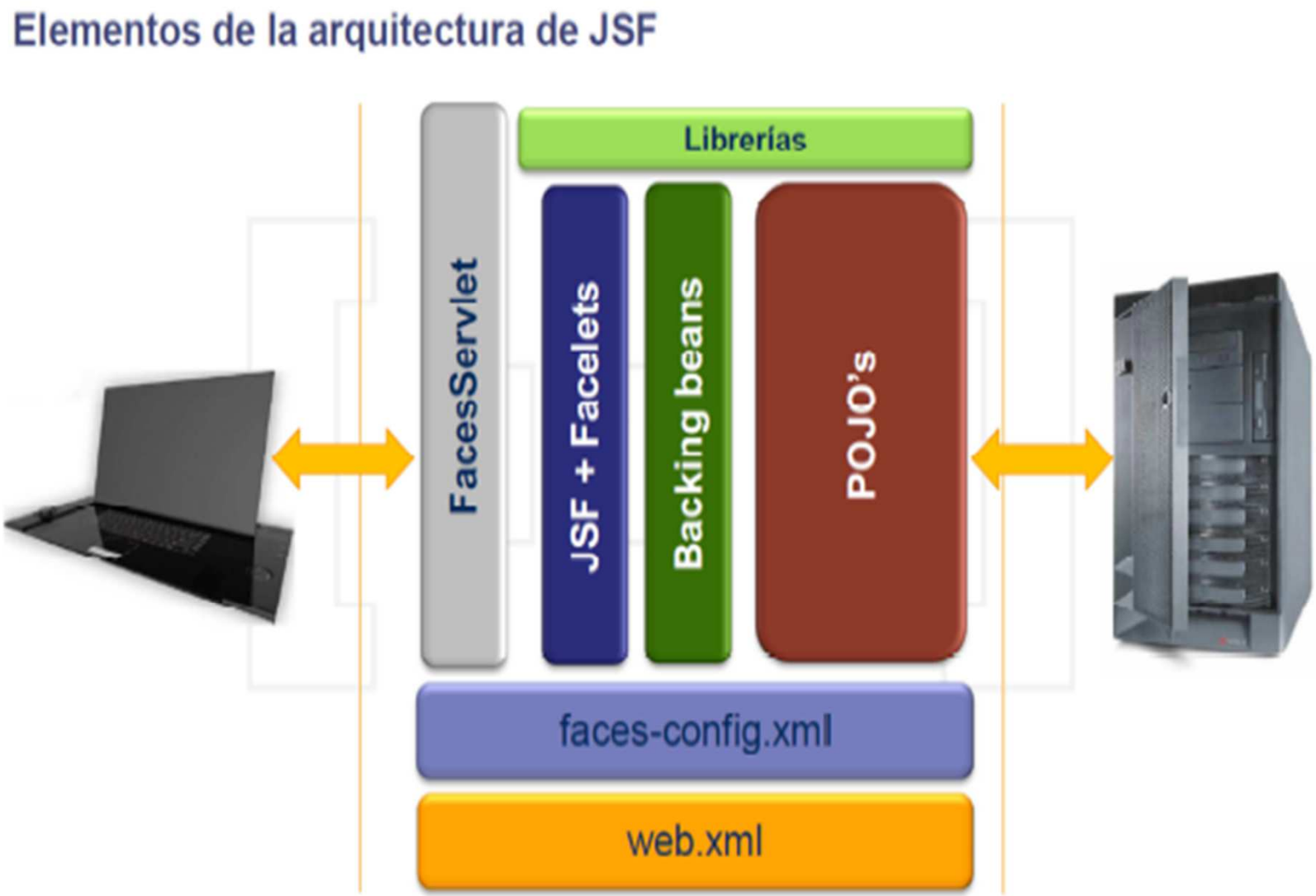
Vistazo general a JSF

- JavaServer Faces amplia el concepto de desarrollo web basado en etiquetas.



- Catálogo estándar que puede ser implementado y da la oportunidad al desarrollador de crear sus propios componentes.

Vistazo general a JSF



Vistazo general a JSF

- Ciclo de desarrollo
  1. Mapeo de la instancia de faces servlet (front controller de jsf).
  2. Definición de parámetros de configuración.
  3. Creación y declaración de los backing beans
  4. Creación de vistas utilizando tags de componentes

Configuración

- El primer paso para poder trabajar con JSF, es declarar el servlet de JSF como Front Controller

```
<servlet>
	<servlet-name>Faces Servlet</servlet-name>
	<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
</servlet>
<servlet-mapping>
	<servlet-name>Faces Servlet</servlet-name>
	<url-pattern>*.xhtml</url-pattern>
</servlet-mapping>
```

Configuración

- El siguiente paso, será configurar todos los parámetros de configuración de forma adecuada.
- Existen distintos parámetros de configuración, que se definen como parámetros de contexto web.

```
<context-param>
      <param-name>javax.faces.PROJECT_STAGE</param-name>
      <param-value>Development</param-value>
</context-param>
```

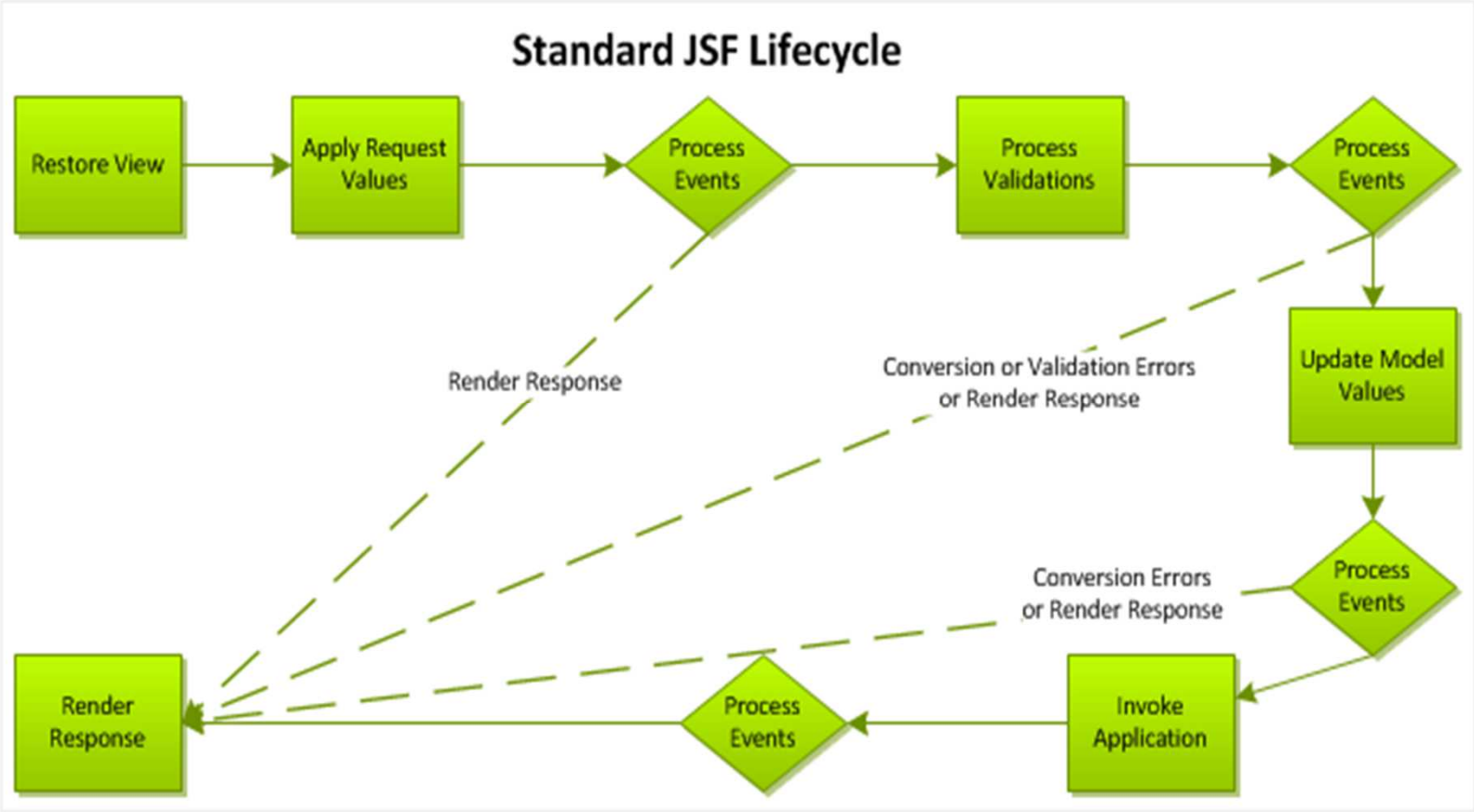
Configuración

- Algunos de los parámetros de configuración mas importantes son:
  - **javax.faces.CONFIG_FILES**. Listado de ficheros separados por comas, con la configuración de JSF, de no existir el por defecto será **"/WEB-INF/faces-config.xml"**
  - **javax.faces.PROJECT_STAGE**. Estado en el que se encuentra el proyecto. Los posibles valores son **Development**, **UnitTest**, **SystemTest** o **Production**.

Configuración

- **javax.faces.STATE_SAVING_METHOD**. Define donde se guarda la información del estado de la vista. Los posibles valores son:
  - **server** (por defecto), que lo almacena en HttpSession.
  - **client**, que lo almacena en un campo oculto de la vista.
- **javax.faces.FACELETS_SKIP_COMMENTS**. Permite indicar que no se rendericen los comentarios.

Fases de una Request



Fases de una Request

- **Restore View**
  - Primera fase.
  - La Request llega al Faces Servlet.
  - Se busca la vista en el almacén de vistas del servidor o se recupera de la petición, y si no está la crea.
  - La vista se materializa en un árbol de componentes.
  - Si se crea el árbol por primera vez, se pasa a la ultima fase **Render Response**.

Fases de una Request

- **Apply Request Values**
  - Aplica los valores de la Request, a los distintos componentes del árbol con los que están asociados (actualizando el árbol).

Fases de una Request

- **Process Validation**
  - En esta fase se aplican los **Validadores** y los **Conversores**.
  - Si se produce un error de validación, se incluye un mensaje en el **FacesContext**, marcándose el componente como invalido y se salta a la fase **Render Response**.

Fases de una Request

- **Update Model Values**
  - Se crean/actualizan los **Managed Beans** con los valores de los componentes del árbol a los cuales están asociados, esta operación, se realiza interpretando las expresiones EL.

Fases de una Request

- **Invoke Application**
  - Se ejecuta el método asociado al evento invocado.
  - El controlador jsf permite la ejecución de la lógica de negocio de la aplicación, ya que están garantizadas las transformaciones y aplicaciones.
  - Se decide el identificador de la vista a mostrar.

Fases de una Request

- **Render Response**
  - Según el identificador de la vista y el **Navigation Handler** definido, se selecciona la vista a renderizar y se renderiza.
  - Se actualiza el árbol de la vista residente en memoria.
  - Muestra la vista en el estado actual.

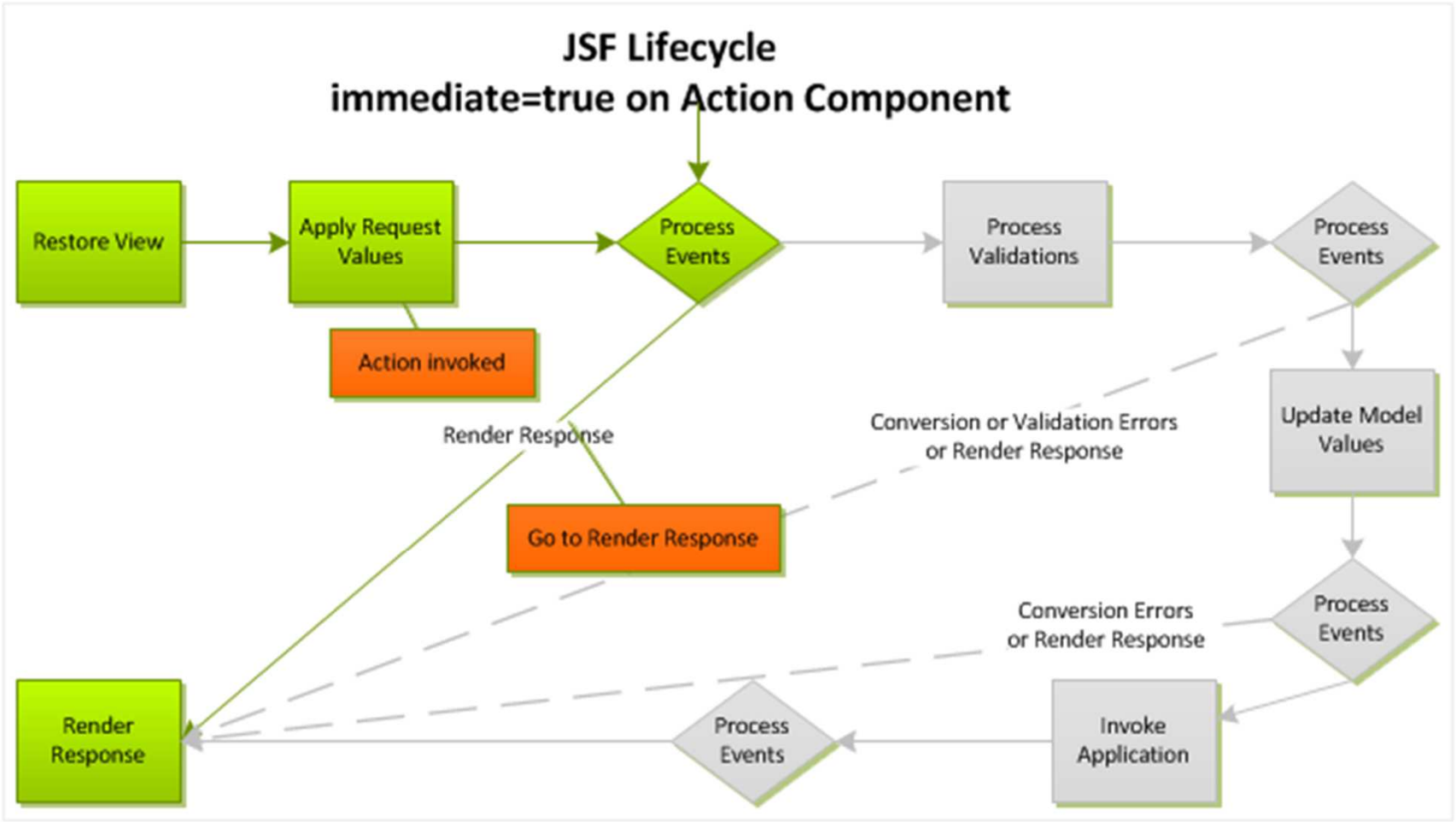
Componentes Immediate

- Se pueden realizar las peticiones al servidor mediante acciones **inmediate**, esto es, aplicando a los componentes **UICommand**, el atributo **inmediate**.

```
<h:commandButton value="Salir" action="salir" immediate="true" />
```

- Con este atributo, se consigue saltar directamente a la fase de **Invoke Application** sin aplicar validaciones y conversiones

Componentes Immediate



Vistas

- Las vistas en JSF, se pueden componer en **jsp** o en **xhtml**.
- Se recomienda el empleo de **xhtml**, dado que JSP, conlleva una compilación a servlet, el cual no se emplea y ocupa espacio en memoria.

Vistas

- Las vistas tienen distintas propiedades, entre las que se encuentran
  - **Value**: Describe el valor del componente, bien con un literal o con una expresión EL que haga referencia a un atributo de un Managed Bean.
  - **Binding**: Expresión EL, que asocia el componente del xhtml, con un atributo de clase de un **ManagedBean**.
  - **Rendered**: Permite definir con una expresión EL, cuando se ha de pintar el componente.

Vistas

- El nodo principal ha de ser **<f:view>** o **<f:body>**.

Librerias De Etiquetas

- Las páginas JSF se construyen con librerías de etiquetas y con el lenguaje de expresiones EL (ej: #{miBean.miPropiedad})
- Las librerías de etiquetas con sus atributos permiten personalizar el aspecto y el comportamiento de cada componente.

Librerias De Etiquetas

- Las librerías de etiquetas estándar son:
  - las **core tags libraries**: definen vistas, listeners, converters, validators, etc.
  - las **html tags libraries**: definen componentes de entrada, de salida, de acción, de selección, de agrupación

Librerias De Etiquetas

- **Core: Vistas**
  - <f:view/>: Crea una vista principal.
  - <f:subview/>: Crea una vista secundaria.
  - <f:facet/>: Añade un facet a un componente.

Librerias De Etiquetas

- **Core: Listeners**
  - <f:actionListener/>: Añade un ActionListener a un componente.
  - <f:valueChangeListener/>: Añade un ValueChangeListener a un componente.
  - <f:setPropertyChangeListener/>: Añade un ActionListener a un componente que establece una propiedad de un bean a un valor dado.

Librerias De Etiquetas

- **Core: Converters**
  - <f:converter/>: Añade un Converter a un componente.
  - <f:convertDateTime/>: Añade un DateTimeConverter a un componente.
  - <f:convertNumber/>: Añade un NumberConverter a un componente.

Librerias De Etiquetas

- **Core: Validators**
  - <f:validator/>: Añade un Validator a un componente.
  - <f:validateDoubleRange/>: Valida que el valor de un componente esté dentro de un rango de valores de tipo double.
  - <f:validateLength/>: Valida la longitud del texto de un componente.
  - <f:validateLongRange/>: Valida que el valor de un componente esté dentro de un rango de valores de tipo long.

Librerias De Etiquetas

- **Core: Otros**
  - <f:attribute/>: Añade un atributo (clave/valor) a un componente.
  - <f:param/>: Añade un parámetro a un componente.
  - <f:loadBundle/>: Carga un ResourceBundle y guarda las propiedades como un Map.
  - <f:selectitems/>: Specifica los items de un select.
  - <f:selectitem/>: Specifica un item de un select.
  - <f:verbatim/>: Añade etiquetas HTML dentro de una página JSF.

Librerias De Etiquetas

- **HTML: Componentes de entrada**
  - <h:inputText/>: Simple línea de texto de entrada.
  - <h:inputTextarea/>: Múltiples líneas de texto de entrada.
  - <h:inputSecret/>: Contraseña de entrada.
  - <h:inputHidden/>: Campo oculto.

Librerias De Etiquetas

- **HTML: Componentes de salida**
  - <h:outputLabel/>: Etiqueta para otro componente.
  - <h:outputLink/>: Enlace HTML.
  - <h:outputFormat/>: Formatea un texto de salida.
  - <h:outputText/>: Simple línea de texto de salida.
  - <h:outputStylesheet/>: Import de un CSS.
  - <h:outputScript/>: Import de un JavaSript.
  - <h:graphicImage/>: Muestra una imagen.
  - <h:message/>: Muestra el mensaje más reciente para un componente.
  - <h:messages/>: Muestra todos los mensajes.

Librerias De Etiquetas

- **HTML: Componentes de acción**
  - <h:commandButton/>: Botón: submit, reset o pushbutton.
  - <h:commandLink/>: Enlace asociado a un botón pushbutton.

Librerias De Etiquetas

- **HTML: Componentes de selección**
  - <h:selecOneListbox/>: Selección simple para lista desplegable.
  - <h:selectOneMenu/>: Selección simple para menú.
  - <h:selectOneRadio/>: Conjunto de botones radio.
  - <h:selectBooleanCheckbox/>: Checkbox.
  - <h:selectManyCheckbox/>: Conjunto de checkboxes.
  - <h:selectManyListbox/>: Selección múltiple de lista desplegable.
  - <h:selectManyMenu/>: Selección múltiple de menu.

Librerias De Etiquetas

- **HTML: Componentes de agrupación**
  - <h:head/>: Cabecera HTML.
  - <h:body/>: Cuerpo HTML.
  - <h:form/>: Formulario HTML.
  - <h:panelGrid/>: Tabla HTML.
  - <h:panelGroup/>: Dos o más componentes que son mostrados como uno.
  - <h:dataTable/>: Tabla de datos.
  - <h:column/>: Columna de un dataTable.

¿Qué Es Un Managed Bean?

- Se trata de objetos manejados por el contexto de JSF, esto es, objetos de los cuales el desarrollador no es responsable.
- Cuando se referencia un **Managed Bean**, el contexto de JSF, creará el objeto, lo inicializará y lo almacenará en el ámbito definido, o lo retornará si ya existe.

¿Qué Es Un Managed Bean?

- Es necesario, que los **Managed Bean**, tengan
  - **Constructor** por defecto.
  - Métodos de **Get** y **Set** para las propiedades.

¿Qué Es Un Managed Bean?

- Para que un objeto sea un **Managed Bean**, hay que declararlo como tal, esto se puede hacer con
  - Anotaciones

```
@ManagedBean(name="user")
@SessionScoped
public class UserBean {...}
```

  - Declarativo en el **faces-config.xml**

```
<managed-bean>
    <managed-bean-name>userBean</managed-bean-name>
    <managed-bean-class>com.examples.UserBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

¿Qué Es Un Managed Bean?

- Como se puede apreciar en los ejemplos, es necesario indicar el ámbito del objeto, sino se indica el por defecto es **Request**.

¿Qué Es Un Managed Bean?

- La declaración por **anotaciones** no es posible emplearla, cuando se definen **Managed Beans**, de los cuales no se tiene el código fuente, para ello solo se puede emplear la declarativa.
- Este es el caso de Los Map, List, Set, …

¿Qué Es Un Managed Bean?

- Declaración de un List

```
<managed-bean>
	<managed-bean-name>newsletters</managed-bean-name>
	<managed-bean-class>java.util.ArrayList</managed-bean-class>
	<managed-bean-scope>application</managed-bean-scope>
	<list-entries>
		<value-class>javax.faces.model.SelectItem</value-class>
		<value>#{newsletter0}</value>
		<value>#{newsletter1}</value>
		<value>#{newsletter2}</value>
		<value>#{newsletter3}</value>
	</list-entries>
</managed-bean>
```

¿Qué Es Un Managed Bean?

- Declaración de un Map

```
<managed-bean>
	<managed-bean-name>newsletters</managed-bean-name>
	<managed-bean-class>java.util.Map</managed-bean-class>
	<managed-bean-scope>application</managed-bean-scope>
	<map-entries>
		<key-class>java.lang.String</key-class>
		<value-class>java.math.BigDecimal</value-class>
		<map-entry>
			<key>My Early Years: Growing Up on *7</key>
			<value>30.75</value>
		</map-entry>
		<map-entry>
			<key>Web Servers for Fun and Profit</key>
			<value>40.75</value>
		</map-entry>
	</map-entries>
</managed-bean>
```

Ambitos De Un Managed Bean

- Tradicionalmente, se pueden almacenar en tres ámbitos (JSF 1.x)
  - **application**.
  - **session**.
  - **request**.
- En JSF 2.0, se amplían a
  - **view**.
  - **none**.

Ambitos De Un Managed Bean

- **Application**
  - Se guarda la información durante toda la vida de la aplicación web, independientemente de todas las peticiones y sesiones que se realicen.
  - Los bean se instancian con la primera petición a la aplicación y desaparece cuando la aplicación web se elimina del servidor.

Ambitos De Un Managed Bean

- **Application**
  - Si queremos que el bean se instancie antes de que se muestre la primera página de la aplicación, usamos la propiedad eager a true.
    - Anotaciones

```
@ManagedBean(eager=true)
```

    - Declarativo

```
<managed-bean eager="true">
```

Ambitos De Un Managed Bean

- **Session**
  - En este ámbito, los bean se guardan desde que el usuario comienza una sesión hasta que ésta termina (porque el tiempo expiró o se invocó al método invalidate sobre un objeto HttpSession).

Ambitos De Un Managed Bean

- **Request**
  - Comienza cuando se envía una petición al servidor y termina cuando se devuelve la respuesta al usuario.
  - Los mensajes de estado y de error que se muestran al usuario son buenos candidatos a ser request, ya que se muestran una vez que el servidor devuelve la respuesta.

Ambitos De Un Managed Bean

- **View**
  - Los bean duran desde que se muestra una página JSF al usuario hasta que el usuario navega hacia otra página.
  - Es muy útil para páginas que usan AJAX.

Ambitos De Un Managed Bean

- **None**
  - Los beans se instancian cuando son necesitados por otros beans, y se eliminan cuando esta necesidad desaparece.

Propiedades Manejadas

- Los **Managed Beans** pueden tener propiedades manejadas, es decir, propiedades, que gestiona el contexto.
- Como para la declaración de los **Managed Bean**, para declaración de **Managed Property**, se pueden emplear
  - **Anotaciones**
  - **faces-config.xml**

Propiedades Manejadas

- Para las anotaciones, se emplea @**ManagedProperty**

```
@ManagedBean
@SessionScoped
public class UserBean {
        @ManagedProperty(value="Ana")
        private String name;
}
```

Propiedades Manejadas

- En el **value** se pueden emplear expresiones EL, para hacer referencia otros **Managed Bean**.

```
@ManagedBean
@SessionScoped
public class UserBean {
        @ManagedProperty(value="#{amigo}")
        private UserBean amigo;
}
```

Propiedades Manejadas

- Los **Bean** a los que referencia deben de ser de un **Ambito** igual o mas longevo, desde **Session** no se puede referenciar a **Request**.

Propiedades Manejadas

- De forma declarativa

```
<managed-bean>
        <managed-bean-name>userBean</managed-bean-name>
        <managed-bean-class>com.examples.UserBean</managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
        <managed-property>
                <property-name>name</property-name>
                <value>Ana</value>
        </managed-property>
</managed-bean>
```

Propiedades Manejadas

- De forma declarativa con referencia otro **Managed Bean**.

```
<managed-bean>
	<managed-bean-name>userBean</managed-bean-name>
	<managed-bean-class>com.examples.UserBean</managed-bean-class>
	<managed-bean-scope>session</managed-bean-scope>
	<managed-property>
		<property-name>amigo</property-name>
		<value>#{amigo}</value>
	</managed-property>
</managed-bean>
```

Propiedades Manejadas

- Si la propiedad es de tipo **Map**, **List**, **Set**, se puede definir los valores directamente en la declaración del **Managed Bean**

```
<managed-bean>
	<managed-bean-name>userBean</managed-bean-name>
	<managed-bean-class>com.examples.UserBean</managed-bean-class>
	<managed-bean-scope>session</managed-bean-scope>
	<managed-property>
		<property-name>colores</property-name>
		<list-entries>
			<value>Rojo</value>
			<value>Verde</value>
			<value>Amarillo</value>
		</list-entries>
	</managed-property>
</managed-bean>
```

Ciclo De Vida Del Managed Bean

- Cuando a un método le asocias la anotación @**PostConstruct** se ejecutará después de la inicialización del bean pero antes de que sea guardado en su scope correspondiente.
- Si al método le asocias la anotación @**PreDestroy** se ejecutará justo antes de que el bean sea eliminado del scope.

Acciones

- Corresponde a la invocación de métodos de los **Managed Bean** o a la invocación de una pagina JSF.
- Se emplea el atributo **action** de las etiquetas **CommandButton** o **CommandLink**.

Acciones

- Para la invocación de un método de un **Managed Bean**.

```
<h:commandButton value="Entrar" action="#{Login.submit}“ />
```

- El método ha de retorna un **string**, que será interpretado como la pagina JSF que se empleará como vista de salida.

```
public String submit() {
        return "login";
}
```

Acciones

- Para la invocación de un método de un **Managed Bean**.

```
<h:commandButton value="Entrar" action="#{Login.submit}“ />
```

- Para la invocación de una pagina JSF

```
<h:commandButton value="Entrar" action=“login“ />
```

- El ejemplo lleva por defecto a la pagina **login.xhtml**

Acciones

- Por defecto, JSF hace **POST** y **forward**.
- La ventaja es que no perdemos la request, pero como desventaja no se actualiza la URL del navegador (eso es a veces confuso para los usuarios, y malo para los bookmarks y el SEO)
- Podemos obligar a que JSF haga un redirect, aunque esto implique perder la request original, y por ende forzar a que si queremos utilizar un **Managed Bean** este tiene tener un **SessionScope**.

```
<h:commandButton label="Login" action"login?faces-redirect=true"/>
```

Acciones

- Una opción mas elegante disponible en JSF 2.0 es el empleo del objeto flash, que permite realizar la redirección, manteniendo el mismo objeto flash, con un ámbito inferior a **Session**, y superior a **Request**.

Acciones

- El empleo en las vistas sería

```
<h:inputText id="nombre" value="#{flash.usuario.nombre}“ required="true" />
```

- Para el empleo en los **Managed Bean** se puede acceder al objeto de dos formas
  - ELFlash.getFlash()
  - FacesContext.getCurrentInstance().getExternalContext().getFlash()

Acciones

- Para hacer GET, se han de emplear **button** y **link** en vez de **commandButton** y **commandLink**.

```
<h:link value="Aceptar" outcome="login?myname=Ana"/>
```

- Para recoger los parámetros de la **request**, en las vistas, habrá que hacer

```
<f:metadata>
        <f:viewParam name="myname" value="#{userBean.name}"/>
</f:metadata>
```

- Donde **myname**, es parámetro de **request**, que se asigna al atributo **name**

Eventos

- Existen dos eventos que se pueden manejar en JSF
  - **ActionListener**
  - **ValueChangeListener**

Eventos

- **ActionListener**
  - Se ejecutan antes que los **Action**.
  - No provocan navegación.
  - Asociado con el atributo de las etiquetas **actionListener**, o con la etiqueta **ActionListener**, que a su vez estará asociada a una clase que implemente **ActionListener**.
  - Se ha de definir en el **Managed Bean**, un método que reciba como parámetro **ActionEvent** y no retorne nada.

Eventos

- **ValueChangeListener**
  - Lanzados por:
    - comboboxes,
    - checkboxes.
    - radiobuttons.
    - textfields.
    - selectonemenu.
    - selectbooleanmenu.
    - selectoneradio.
    - inputtext.

Eventos

- **ValueChangeListener**
  - No producen un **submit** automático, hay que forzarlo con javascript.

```
<h:selectOneMenu
        value="#{controladorPersona.persona.ciudad}"
        onchange="submit()“
        valueChangeListener="#{controladorPersona.cambioCiudad}">
                <f:selectItems value="#{controladorPersona.ciudades}" />
</h:selectOneMenu>
```

Eventos

- **ValueChangeListener**
  - En el **Managed Bean**, se ha de definir un método que recibirá un objeto **ValueChangeEvent**.

```
public void cambioCiudad(ValueChangeEvent e){
        persona.setCiudad(e.getNewValue().toString());
}
```

Contexto

- Se puede acceder de forma programática al contexto de JSF, empleando.

```
FacesContext.getCurrentInstante();
```

- El contexto se encarga de crear e inicializar los bean para los distintos ámbitos, pero no se encarga de su destrucción, excepto que se elimine el ámbito. Para destruir un bean:

```
FacesContext.getCurrentInstante().getExternaContext().getSessionMap().remove(“”);
```

Contexto

- Se puede acceder a un **Managed Bean** desde el código empleando el contexto.

```
FacesContext.getCurrentInstance().getApplication()
        .evaluateExpressionGet(context, "#{myBean}", MyBean.class);
```

- Y también acceder a la **HttpRequest**.

```
FacesContext.getCurrentInstance().getExternalContext()
        .getRequest();
```

- Y a la **HttpSession**

```
FacesContext.getCurrentInstance().getExternalContext()
        .getSession(true);
```

Contexto

- Y los parámetros de la **HttpRequest**

```
FacesContext.getCurrentInstance().getExternalContext()
        .getRequestParameterMap().get("id");
```

- Y los atributos de la **HttpSession**.

```
FacesContext.getCurrentInstance().getExternalContext().getSessionMap();
```

- Y los atributos de la **HttpRequest**.

```
FacesContext.getCurrentInstance().getExternalContext().getRequestMap();
```

Contexto

- También el contexto, permite resolver expresiones EL, con las que acceder a los **Managed Bean**, a través del objeto **ELContext**.

```
FacesContext ctx = FacesContext.getCurrentInstance();

ELContext ec = ctx.getELContext( );
```

- Para ello

```
Application app = ec.getApplication( );

UserBean user = (UserBean) app.evaluateValueExpressionGet(
                                                    ctx,"#{userBean}",UserBean.class);
```

Contexto

- O también con un **ExpressionFactory**

```
ExpressionFactory ef = app.getExpressionFactory( );

ValueExpression ve =
                        ef.createValueExpression(ec,"#{userBean}",UserBean.class);

UserBean user = (UserBean) ve.getValue(ec);

user.setName(“Ana");
```

Lenguaje de Expresiones (EL)

- Creado originalmente como parte del jstl (java standard tag library 1.0)
- Se traslada a JSF, empleando el operador **#**, en lugar del operador **$**.
- Son bidireccionales (lectura y escritura) leen o actualizan valores.
- Pueden referenciar métodos.

Lenguaje de Expresiones (EL)

- Se definen una serie de objetos implícitos en el ámbito de las expresiones EL, estos son
  - #{**application**}: el ServletContext.
  - #{**facesContext**}: el FacesContext actual.
  - #{**session**}: el HttpSession actual.
  - #{**request**}: el HttpServletRequest actual.
  - #{**view**}: el UIViewRoot actual.
  - #{**component**}: el UIComponent actual.

Lenguaje de Expresiones (EL)

- #{**applicationScope**}: un map con los atributos del scope aplication actual.
- #{**sessionScope**}: un map con los atributos de la session actual.
- #{**requestScope**}: un map con los atributos de la request actual.
- #{**viewScope**}: un map con los atributos de la view actual.
- #{**initParam**}: un map con los parámetros del context actual.
- #{**param**}: un map con los parámetros de la request actual.

Lenguaje de Expresiones (EL)

- #{**paramValues**}: un map con los valores de los parámetros de la request actual.
- #{**header**}: un map con la cabecera de la request actual.
- #{**headerValues**}: un map con los valores de la cabecera de la request actual.
- #{**cookie**}: un map con los atributos de la cookie.

Lenguaje de Expresiones (EL)

- Cuando se referencia aun objeto no implícitos con expresiones EL, este se busca siguiendo el orden
  - Request
  - View
  - Session
  - Application

Lenguaje de Expresiones (EL)

- Para acceder a las propiedades y a los métodos de los beans.

```
<h:outputText value="#{userBean.profile}"/>
```

- Podemos utilizar operadores aritméticos, lógicos o comprobar si el valor es empty.

```
<h:outputText rendered="#{userBean.profile=='VIP'}" value="Bono regalo de
#{bono.base-10} Euros"/>
```

- Podemos invocar a métodos con parámetros, siempre que no estén sobrecargados.

```
<h:commandButton value="Aceptar" action="#{userBean.addText('texto')}"/>
```

Lenguaje de Expresiones (EL)

- También se puede acceder a los **Managed Bean** desde el código java

```
FacesContext ctx = FacesContext.getCurrentInstance();
Application app = ctx.getApplication( );
ELContext ec = ctx.getELContext( );
ExpressionFactory ef = app.getExpressionFactory( );

String name = (String)
        app.evaluateValueExpressionGet(ctx,"#{userBean.name}",String.class);

ValueExpression ve =
        ef.createValueExpression(ec,"#{userBean.name}",String.class);

name = (String) ve.getValue(ec);
ve.setValue(ec, "Serena");

ve =  ef.createValueExpression(ec,"#{userBean}",UserBean.class);

UserBean ub = (UserBean) ve.getValue(ec);
ub.setName("Serena");
```

Lenguaje de Expresiones (EL)

Example	Description
`#{myBean.value}`	Returns the `value` property of the object stored under the key `myBean`, or the element stored under the key `value` if `myBean` is a `Map`.
`#{myBean['value']}`	Same as `"{#myBean.value}"`.
`#{myArrayList[5]}`	Returns the fifth element of a `List` stored under the key `myArrayList`.
`#{myMap['foo']}`	Returns the object stored under the key `foo` from the `Map` stored under the key `myMap`.
`#{myMap[foo.bar]}`	Returns the object stored under the key that equals the value of the expression `foo.bar` from the `Map` stored under the key `myMap`.
`#{myMap['foo'].value}`	Returns the `value` property of the object stored under the key `foo` from the `Map` stored under the key `myMap`.
`#{myMap['foo'].value[5]}`	Returns the fifth element of the `List` or array stored under the key `foo` from the `Map` stored under the key `myMap`.
`#{myString}`	Returns the `String` object stored under the key `myString`.
`#{myInteger}`	Returns the `Integer` object stored under the key `myInteger`.
`#{user.role == 'normal'}`	Returns `true` if the `role` property of the object stored under the key `user` equals normal. Returns `false` otherwise.
`#{(user.balance - 200) == 0}`	If the value of the `balance` property of the object stored under the key user minus 200 equals zero, returns `true`. Returns `false` otherwise.
`Hello #{user.name}!`	Returns the string "Hello" followed by the `name` property of the object stored under the key `user`. So if the user's name is Sean, this would return "Hello Sean!"
`You are #{(user.balance > 100) ? 'loaded' : 'not loaded'}`	Returns the string "You are loaded" if the `balance` property of the object stored under the key `user` is greater than 100; returns "You are not loaded" otherwise.
`#{myBean.methodName}`	Returns the method called `method` of the object stored under the key `myBean`.
`#{20 + 3}`	Returns 23.

Lenguaje de Expresiones (EL)

Syntax	Alternative	Operation
.		Access a bean property, method, or `Map` entry
[]		Access an array or `List` element, or `Map` entry
()		Creates a subexpressions and controls evaluation order
? :		Conditional expression: `ifCondition ? trueValue : falseValue`
+		Addition
-		Subtraction and negative numbers
*		Multiplication
/	div	Division
%	mod	Modulo (remainder)
==	eq	Equals (for objects, uses the `equals()` method)
!=	ne	Not equal
<	lt	Less than
>	gt	Greater than
<=	le	Less than or equal
>=	ge	Greater than or equal
&&	and	Logical AND
||	or	Logical OR
!	not	Logical NOT
empty		Tests for an empty value (`null`, an empty `String`, or an array, `Map` or `Collection` with no values)

Reglas De Navegación

- Existe un componente que se encarga de resolver la vista a mostrar tras una petición, este componente se llama **NavigationHandler**.
- Por defecto el **NavigationHandler** resolverá el **String** definido en el atributo **Action** o el retornado por un **método de Acción**, como una url a partir de **WebContent**, añadiendo la extensión **xhtml**.

Reglas De Navegación

- También, se pueden definir reglas de navegación mas granulares en el fichero **faces-config.xml**.
- Para ello se tienen los **Navigation Case** y los **Navigation Rules**.

Reglas De Navegación

- Un ejemplo de regla de navegación

```
<navigation-rule>
	<from-view-id>/index.xhtml</from-view-id>
	<navigation-case>
		<from-outcome>login</from-outcome>
		<to-view-id>/login.xhtml</to-view-id>
	</navigation-case>
	<navigation-case>
		<from-outcome>register</from-outcome>
		<to-view-id>/register.xhtml</to-view-id>
	</navigation-case>
</navigation-rule>
```

Reglas De Navegación

- En la anterior regla de navegación podemos ver que desde la página **index.xhtml** podemos ir a **login.xhtml** o a **register.xhtml** dependiendo si el **método de accion** devuelve "**login**" o "**register**" respectivamente.

Reglas De Navegación

- También, se puede definir de que método de acción tiene que venir la navegación

```
<navigation-rule>
	<from-view-id>from_page.xhtml</from-view-id>
	<navigation-case>
		<from-action>#{ManagedBean.actionMethod}</from-action>
		<from-outcome>condition 1</from-outcome>
		<to-view-id>/to_page1.xhtml</to-view-id>
	</navigation-case>
</navigation-rule>
```

Reglas De Navegación

- Se pueden definir condiciones que se han de cumplir para aplicar la regla de navegación

```
<navigation-case>
    <from-outcome>login</from-outcome>
    <if>#{userBean.profile != 1}</if>
    <to-view-id>/login.xhtml</to-view-id>
</navigation-case>
```

Reglas De Navegación

- También realizar una redirección para mostrar la url de la pagina destino

```
<navigation-case>
    <from-outcome>login</from-outcome>
    <to-view-id>/page2.xhtml</to-view-id>
    <redirect/>
</navigation-case>
```

Formularios

- El acceso a los elementos de un formulario definido con JSF desde Javascript, sigue la siguiente pauta.
  - Los Id de los componentes de los formularios, se construyen como sigue

```
idFormulario:idNodo
```

  - Por lo que el acceso se hará

```
document.getElementById("idFormulario:idNodo")
```

AJAX

- AJAX persigue sustituir las recargas completas de las paginas producidas por peticiones postback, por recargas parciales de los componentes deseados.
- El soporte de Ajax, se basa en el tag <**f:ajax**>, que asociado a un componente con un evento definido.

AJAX

- El tag <**f:ajax**>, tiene dos parámetros
  - **Render** – Los elementos a actualizar con la respuesta.
  - **Execute** – Los datos a enviar al servidor en la petición para operar.

AJAX

- En ambos atributos, se pueden poner los ID separados por espacios, de los componentes afectados, o bien recurrir a las siguientes constantes.
  - **@all** – Opera con toda la vista
  - **@form** – Opera con el formulario donde este la etiqueta.
  - **@this** – Opera con el elemento que provoca el evento.
  - **@none** – No opera con datos.

AJAX

- Un ejemplo

```
<h:form>
      <h:inputText value="#{bankingBeanAjax.customerId}"/>
      <h:inputSecret value="#{bankingBeanAjax.password}"/>
      <h:commandButton value="Show Current Balance"
                              action="#{bankingBeanAjax.showBalance}">
            <f:ajax execute="@form“ render="ajaxMessage1"/>
      </h:commandButton>
      <h:outputText value="#{bankingBeanAjax.message}"
                              id="ajaxMessage1"/>
</h:form>
```

Conversores

- Un conversor se utiliza para obligar a que la entrada de un dato, se realice en un formato predefinido, de no cumplirse dicha premisa, se lanzará un error de conversión.
- Convierten el String que introduce el usuario en un objeto java, y viceversa.

Conversores

- Se pueden aplicar convertidores a las siguientes etiquetas
  - <h:outputText>
  - <h:outputFormat>
  - <h:outputLink>
  - <h:outputLabel>
  - <h:inputText>
  - <h:inputTextarea>
  - <h:inputHidden>
  - <h:inputSecret>
  - <h:selectBooleanCheckbox>
  - <h:selectManyListbox>
  - <h:selectMaynyMenu>
  - <h:selectOneRadio>
  - <h:selectOneListbox>
  - <h:selectOneMenu>

Conversores

- Se pueden asociar los conversores a los campos de las siguientes maneras
  - Con la propiedad converter de las etiquetas

```
<h:outputText value="#{myBean.date}" converter="myConverter">
```

  - Añadir la etiqueta converter

```
<h:outputText value="#{myBean.date}">
      <f:converter converterId="myConverter"/>
</ h: outputText>
```

Conversores

- Además se puede añadir un mensaje personalizado en caso de no cumplir la conversión

```
<h:inputText id="quantity" value="#{item.quantity}"
             converterMessage="Un entero por favor!">
      <f:convertNumber type="number" integerOnly=“true"/>
</h:inputText>
```

- Existen dos conversores predefinidos
  - **ConvertDateTime**
  - **ConvertNumber**

Conversores

- **ConvertDateTime**

```
<h:outputText value="#{myBean.date}">
      <f:convertDateTime type="date" dateStyle="medium"/>
</h:outputText>
```

Conversores

- **ConvertDateTime**
  - Que tiene los siguientes atributos

Nombre del atributo	Descripción
datestyle	Especifica el estilo de formato para la porción de la fecha de la cadena. Las opciones válidas son de short, medium (por defecto), long y full. Sólo es válido si se establece el tipo de atributo.
timeStyle	Especifica el estilo de formato para la porción de tiempo de la cadena. Las opciones válidas son de short, medium (por defecto), long y full. Sólo es válido si se establece el tipo de atributo.
timezone	Especifica la zona horaria para la fecha. Si no se establece, hora del meridiano de Greenwich (GMT) se utilizará.
locale	El idioma local a utilizar para la visualización de esta fecha. Reemplaza la localización actual
pattern	El modelo de formato de fecha utilizado para convertir esta fecha. Utilice este o el tipo de propiedad.
type	Especifica si se debe mostrar la fecha (date), hora (time) o ambas (both).

Conversores

- **ConvertNumber**

```
<h:outputText value="#{myBean.date}">
      <f:convertNumber type="number" maxIntegerDigits="3"/>
</ h: outputText>
```

Conversores

- **ConvertNumber**
  - Que tiene los siguientes atributos

CurrencyCode	Especifica un período de tres dígitos del código de moneda internacional cuando el atributo tipo es la moneda. Utilice este o CurrencySymbol.
CurrencySymbol	Especifica un símbolo específico, como "$", que se utiliza cuando el tipo de atributo es moneda. Utilice este o CurrencyCode.
groupingUsed	True si un símbolo de agrupación, como "," o "" debe ser utilizado. El valor predeterminado es true.
integerOnly	Verdadero si sólo la parte entera del valor de entrada debe ser procesado (todos los decimales será ignorado). El valor predeterminado es falso.
locale	El local que se utilizará para la visualización de este número. Reemplaza el usuario localización actual
minFractionDigits	Una cantidad mínima de decimales que se vea.
maxFractionDigits	Máxima número de decimales que se vea.
minIntegerDigits	Una cantidad mínima de dígitos enteros para mostrar.
maxIntegerDigits	Máxima número de dígitos enteros para mostrar.
pattern	El modelo de formato decimal para convertir este número. Utilice este o tipo de atributo.
tipo	El tipo de número, por el número (number, por defecto), la moneda (currency), o por ciento (percent). Usar este o el patrón de este atributo.

Conversores

- También se pueden definir conversores personalizados, para ello
  - Se ha de crear un implementación de la interface **javax.faces.Converter**, que obliga a implementar los métodos
    - **getAsObject**. Recibe como parámetro el String introducido en el campo, y ha de retornar un Objeto.
    - **getAsString**. Recibe como parámetro el Objeto y ha de retornar el String a introducir en el campo.

Conversores

- De ir mal la conversión, se ha de lanzar una excepción de tipo **ConverterException**, con un parametron **FacesMessage**.

```
FacesMessage msg =
      new FacesMessage(FacesMessage.SEVERITY_ERROR,
      "NIF Conversion error.", "Formato no valido para el NIF.");
throw new ConverterException(msg);
```

Conversores

- Ejemplo de conversor de NIF

```
public Object getAsObject(FacesContext arg0, UIComponent arg1,
String arg2) {
	if (arg2 == null || arg2.equals("")) {
		return null;
	}
	Pattern patron = Pattern.compile("\\d{8}-[A-z]");
	Matcher matcher = patron.matcher(arg2);
	if (!matcher.matches()) {
		FacesMessage msg = new
			FacesMessage(FacesMessage.SEVERITY_ERROR,
			"NIF Conversion error.", "Formato no valido para el
NIF.");
		throw new ConverterException(msg);
	}
	return new Nif(Integer.parseInt(arg2.substring(0, 8)),
							arg2.charAt(9));
}
```

Conversores

- Ejemplo de conversor de NIF

```
public String getAsString(FacesContext arg0, UIComponent arg1,
Object arg2) {
	return arg2.toString();
}
```

Conversores

- Se ha de declarar dicha clase como **Converter**, siempre definiendo un ID, para ello se puede hacer con
  - Anotaciones

```
@FacesConverter(value = "CCNumberConverter")
```

  - Declarativo en **faces-config.xml**

```
<converter>
      <converter-id>CCNumberConverter</converter-id>
      <converter-class>
            curso.converters.CCNumberConverter
      </converter-class>
</converter>
```

Conversores

- Para emplear el conversor personalizado, se empleará la etiqueta converter.

```
<f:converter converterId="CCNumberConverter"/>
```

Validadores

- Permiten comprobar que el dato introducido por el usuario cumple con unas condiciones.
- Con JSF se pueden aplicar cuatro tipos de validaciones
  - Asociada a los componentes
  - Genéricos o programática.
  - Validación por métodos de Managed Beans
  - Componentes personalizados

Validadores

- Se incluyen tres componentes validadores
  - **validateDoubleRange**: Se valida que una entrada numérica está dentro de un rango determinado. Es aplicables a los valores que se pueden convertir a un doble.
  - **validateLength**: Se valida que la longitud de la cadena de entrada está dentro de un rango determinado.
  - **validateLongRange**: Se valida que una entrada numérica está dentro de un rango determinado. Es aplicables a los valores que se pueden convertir a un long.

Validadores

- El uso es similar al de los Conversores

```
<h:inputText id="quantity" value="#{item.quantity}">
      <f:validateLongRange minimum="1"/>
</h:inputText>
```

- Adicionalmente se pueden añadir los mensajes a mostrar en caso de no cumplir la validación

```
<h:inputText id="quantity" value="#{item.quantity}"
            validatorMessage="Minimo uno!">
      <f:validateLongRange minimum="1"/>
</h:inputText>
```

Validadores

- Las validaciones genéricas, son aquellas que no se ciñen a un campo en concreto, y que se pasan antes de realizar la lógica.
- Para ello, se añade al **FacesContext** un **FacesMessage**.

```
FacesContext ctxt = FacesContext.getCurrentInstance();

FacesMessage mess = new FacesMessage();
mess.setSeverity(FacesMessage.SEVERITY_ERROR);
mess.setSummary("Este es el mensaje de error principal");
mess.setDetail("Este es el detalle");

ctxt.addMessage(null, mess);
```

Validadores

- Cuando se incumpla, las validaciones aplicadas en un método de Acción, no es necesario lanzar una excepción, únicamente retornar **null**, para mantener la navegación en la misma pagina, e incluir en la pagina la etiqueta que permite visualizar los errores de validación.

```
<h:messages globalOnly="true" styleClass="error"/>
```

Validadores

- Puede ser interesante si se aplican varias validaciones en un mismo método, comprobar si se ha añadido algún mensaje de validación para no realizar la navegación, para ello

```
if (context.getMessageList().size() > 0) {
        return null;
}
```

Validadores

- También se puede definir un método en el **Managed Bean**, que realice la validación del mismo, en este caso tampoco hay que lanzar excepciones, únicamente añadir los mensajes de validación al contexto.

```
<h:inputText id="ccexpiry" value="#{Usuario.apellido}"
      rendered="true" validator="#{Usuario.validateCCExpiry}">
</h:inputText>
<h:message for="ccexpiry" errorClass="error" />
```

Validadores

- Por ultimo se pueden definir validadores personalizados de forma análoga a como se definen los Conversores personalizados, para ello
  - Se ha de crear una implementación de la interface **javax.faces.validator.Validator**.
    - **Validate**. Método que implementa la lógica de validación, en caso de error se ha de lanzar un **ValidatorException**, con un **FacesMessage**.

Validadores

- Se ha de declarar la clase como validador, para ello se puede hacer con
  - Anotaciones

```
@FacesValidator(value="emailValidator")
```

  - Declarativa en el **faces-config.xml**

```
<validator>
    <validator-id>emailValidator</validator-id>
    <validator-class>com.ejemplo.EmailValidator</validator-class>
</validator>
```

Validadores

- Una vez definido para emplearlo, se usa la etiqueta **validator** que hace referencia al ID

```
<f:validator validatorId="emailValidator"/>
```

i18n

- Mecanismo estándar para la traducción de páginas sin generar contenidos distintos.
- Soporte para múltiples “locales”.

i18n

- Se debe configurar un **Default Locale** y los **Supported Locales**.

```
<faces-config>
  <application><locale-config>
      <default-locale>es</default-locale>
      <supported-locale>en</supported-locale></locale-config>
  </application>
```

i18n

- El **Locale** empleado, se puede recuperar de forma programática de la siguiente forma

```
FacesContext.getCurrentInstance().getViewRoot().getLocale();
```

- También se puede modificar

```
Locale locale = new Locale("es");
FacesContext.getCurrentInstance().getViewRoot().setLocale(locale);
```

i18n

- Para el uso de mensajes traducidos, se deberán definir ficheros **properties** con los mensajes traducidos.
- Se pueden parametrizar los mensajes, empleando la sintaxis **{0}**

i18n

- A modo de recordatorio, los properties, contienen pares clave-valor, donde cada propertie contiene el mismo número de elementos que han sido traducido al locale específico.
  - Objeto para el locale es: MY_MESSAGE=Mi mensaje
  - Objeto para el locale en: MY_MESSAGE=My message

i18n

- La definición del objeto establece la siguiente forma de localizar un recurso Resource Bundle, que en principio será un fichero cuyo nombre estará compuesto del siguiente modo:
  - baseName + "_" + language + "_" + country + "_" + variant

i18n

- Los properties se han de cargar como **Boundles** bien en las Vistas.

```
<f:loadBundle basename=”custom.MyMessages” var=”msg” />
```

- O bien en el faces-config.xml

```
<application>
      <resource-bundle>
            <base-name>custom.MyMessages</base-name>
            <var>msg</var>
      </resource-bundle>
</application>
```

i18n

- Y luego empleados en las Vistas con expresiones EL

```
<h:outputText value="#{msg.mymessage}" />
```

- En el anterior ejemplo, han de existir ficheros
  - custom.MyMessages_es.properties
  - custom.MyMessages_en.properties

i18n

- También se puede acceder al **boundle** desde código.

```
FacesContext facesContext = FacesContext.getCurrentInstance();
String messageBundleName =
        facesContext.getApplication().getMessageBundle();
Locale locale = facesContext.getViewRoot().getLocale();
ResourceBundle bundle =
        ResourceBundle.getBundle(messageBundleName, locale);
```

- O incluso inyectarlo en un ManagedBean

```
@ManagedProperty("#{msg}")
private ResourceBundle bundle;
```

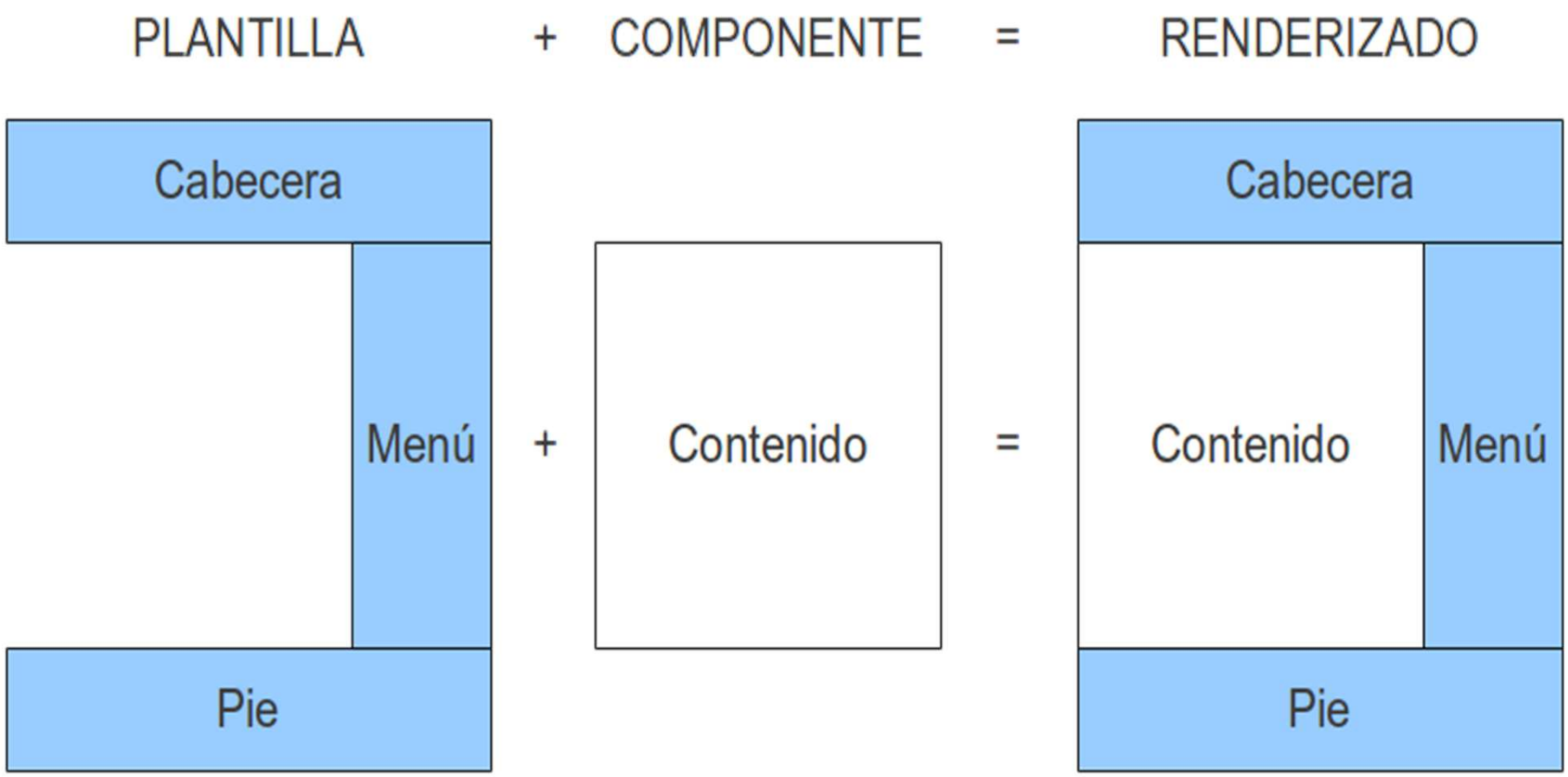
Facelets

- Permiten generar las vistas empleando plantillas.
- Dispone de un **namespace** para las etiquetas

```
xmlns:ui="http://java.sun.com/jsf/facelets"
```

- La plantilla se define como una vista normal, definiendo huecos en los que se podrán insertar elementos que varían.

Facelets


PLANTILLA
+
COMPONENTE
=
RENDERIZADO
Cabecera
Menú
Pie
+
Contenido
=
Cabecera
Contenido
Menú
Pie

Facelets

- En la plantilla, los huecos, serán definidos con la etiqueta.

```
<ui:insert name="head">
```

- Las paginas que empleen la plantilla, tendrán como nodo principal

```
<ui:composition template="/plantilla.xhtml">
```

- Y definirán el relleno de los huecos con

```
<ui:define name="head">
```

Facelets

- O también con

```
<ui:include src=“cabecera.xhtml”>
```

- Que permite referencia a ficheros **xhtml**, a los cuales puede pasarles parámetros.

```
<ui:param name="title" value=“Titulo de la aplicacion" />
```

- Que pueden ser recogidos en el otro fichero empleando expresiones EL.

```
#{title}
```

Facelets

- Se ha de definir un nuevo **ViewHandler** en el contexto de JSF, para ello en el **faces-config.xml**

```
<application>
      <view-handler>
            com.sun.facelets.FaceletViewHandler
      </view-handler>
</application>
```

- Las plantillas se suelen definir dentro de WEB-INF.

Componentes personalizados

- Con JSF 2.0 se pueden crear componentes personalizados, a partir de componentes ya existentes, de una forma sencilla.
- Se dispone de un namespace

```
xmlns:composite="http://java.sun.com/jsf/composite"
```

- Dentro del componente se puede tener acceso al propio componente con la variable “cc”.

Componentes personalizados

- El componente se crea como una Vista, definiendo los siguientes apartados.
  - **composite:interface**. declara el contrato del componente.
  - **composite:implementation**. define la implementación del componente.

Componentes personalizados

- Dentro del nodo **composite:interface**, se definen los atributos
  - **composite:attribute**. declara un atributo en el contrato del componente (la interface).

```
<composite:interface>
      <composite:attribute name="actionLabel" />
      <composite:attribute name="actionMethod"
                           method-signature="java.lang.String action()" />
      <composite:attribute name="resetLabel" />
</composite:interface>
```

Componentes personalizados

- Se puede observar en el ejemplo, como si se desea pasar la referencia a un método, se ha de definir la firma del método a recibir en el atributo **method-signature**.

```
<composite:attribute name="actionMethod"
                     method-signature="java.lang.String action()" />
```

Componentes personalizados

- Dentro del nodo **composite:implementation**, se puede acceder a los atributos a través de la variable "**cc**"

```
<composite:implementation>
      <h:commandButton value="#{cc.attrs.actionLabel}"
            action="#{cc.attrs.actionMethod}" />
      <h:commandButton value="#{cc.attrs.resetLabel}"
            type="reset" />
</composite:implementation>
```

Componentes personalizados

- La definición del componente, se puede situar en la jerarquía de directorios que se desee, siempre que comience en la carpeta **resources** de **WebContent**.
- Para emplear el componente se empleará el **namespace**

```
http://java.sun.com/jsf/composite/<jerarquia de directorios>
```

FACELETS:COMPONENTES

- Para ello se dispone de las siguientes etiquetas:
  - **composite:insertChildren**: sustituye a la etiqueta **ui:insert**.
  - **composite:valueHolder**: permite exponer las propiedades y eventos de los componentes ValueHolder para que sean asignados desde el cliente.

FACELETS:COMPONENTES

- Para ello se dispone de las siguientes etiquetas:
  - **composite:editableValueHolder**: permite exponer las propiedades y eventos de los componentes que implementan la interfaz **EditableValueHolder** para que sean asignados desde el cliente.

FACELETS:COMPONENTES

- Para ello se dispone de las siguientes etiquetas:
  - **composite:actionSource**: permite exponer las propiedades y eventos de los componentes que implementan la interfaz **ActionSource** para que sean asignados desde el client.

Integración con Spring

- La integración entre JSF y Spring, consiste, en delegar la responsabilidad de la creación de los **Managed Bean**, al contenedor de Spring, de tal forma que ahora los **Managed Bean**, no serán bean de JSF, sino de Spring.

Integración con Spring

- Para ello, se necesitan las dependencias con **Spring-web**.

```
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-web</artifactId>
	<version>3.2.3.RELEASE</version>
</dependency>
```

Integración con Spring

- Además, habrá que definir el **ContextLoaderListener** de Spring, para que cargue el contexto de Spring junto con el contexto web.

```
<listener>
	<listener-class>
		org.springframework.web.context.ContextLoaderListener
	</listener-class>
</listener>
```

Integración con Spring

- Con esta configuración, están los dos contenedores a la vez, para que JSF emplee el contenedor de Spring para buscar los **Managed Bean**, habrá que sustituir el componente que se encarga de crear los beans, este es el **ElResolver**, que será sustituido por **SpringBeanFacesELResolver**.

Integración con Spring

- Para hacerlo se incluye en el **faces-config.xml**

```
<application>
      <el-resolver>
      org.springframework.web.jsf.el.SpringBeanFacesELResolver
      </el-resolver>
</application>
```

Integración con Spring

- A partir de este punto, se definirán los bean como beans de Spring, y se podrá acceder a ellos con las expresiones EL desde la vista, como se hace con los beans de JSF.

@VictorHerrero1

Víctor Herrero Cazurro

victorherrerocazurro

victorherrerocazurro