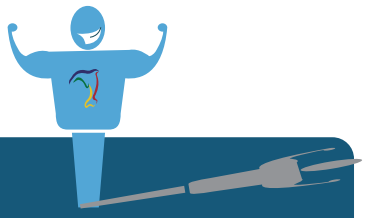
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

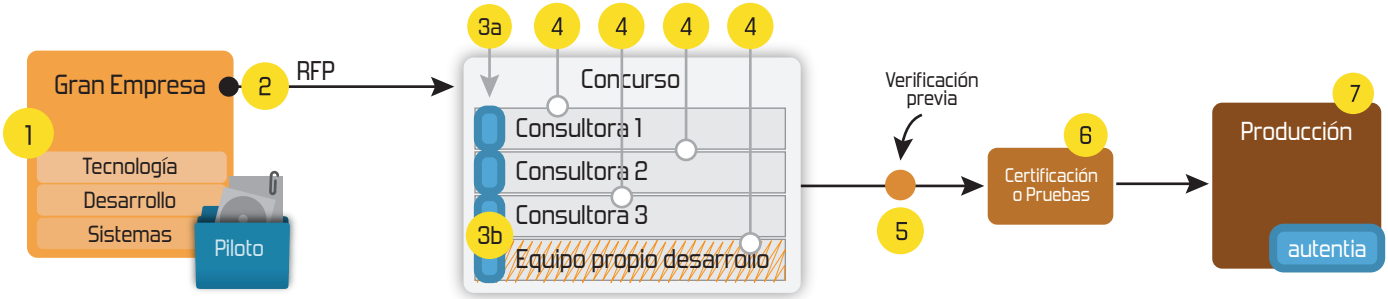
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

E-mail:

Contraseña:

Entrar

Deseo registrarm
He olvidado mis datos de acceso

Últimas Noticias

Pirineos on Tour

VII Autentia Cycling Day

Autentia patrocina la charla sobre Java SE 7 en Madrid

Alfresco Day 2011

XVII Charla Autentia - Grails - Vídeos y Material

Histórico de NOTICIAS

Últimos Tutoriales

Uso de las anotaciones @Embeddable, @Embedded, @AttributeOverrides, @AssociationOverrides

Instalación y uso del plugin de comentarios de Facebook en nuestra Web

10 Nuevas Características de Maven 3

DESARROLLADO POR:
César López de Felipe Abad

Consultor tecnológico de desarrollo de proyectos informáticos.

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/JEE

Fecha de publicación del tutorial: 2011-08-30

Creación de un componente en JSF2.

0. Índice de contenidos.

1. Introducción

Ya vimos hace un tiempo en un tutorial de Alejandro http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=jsfComponent como crear un componente en JSF. Aunque ahora mismo existen muchas librerias de componentes, en ocasiones podrían no ofrecernos el componente que necesitamos y podríamos tener la necesidad de crearlo nosotros.

En este tutorial crearemos un componente personalizado con JSF 2 que básicamente realizará las funciones de un h:outputLabel. Primero crearemos las clases de java, el taglib y la vista y luego le añadiremos el parametro al componente.

2. Entorno.

El tutorial está escrito usando el siguiente entorno:

- Hardware: MacBookPro8,2 (2 GHz Intel Core i7, 4GB DDR3 SDRAM).
- AMD Radeon HD 6490M 256MB.
- Sistema Operativo: Mac OS X Snow Leopard 10.6.7.
- Eclipse Helios

3. Creación de la clase java, el taglib y la vista

.

Para este ejemplo todo lo que necesitaremos será una clase de java para nuestro componente, que tendrá la funcionalidad. Y un archivo xml que será el descriptor de la libreria de tags.

No tenemos necesidad de crear una clase Renderer para nuestro componente. En el ejemplo el código que se encargará de renderizar estará dentro de la clase de java.

El código de la clase java sería el siguiente:

```
01 package com.autentia.jsf.showcase.component;
02
03 import java.io.IOException;
04
05 import javax.faces.component.FacesComponent;
06 import javax.faces.component.UIComponentBase;
07 import javax.faces.context.FacesContext;
08 import javax.faces.context.ResponseWriter;
09
10 @FacesComponent(value = "HtmlCustomComponent")
11 public class HtmlCustomComponent extends UIComponentBase {
12
13     @Override
14     public String getFamily() {
15         return null;
16     }
17
18     @Override
19     public void encodeAll(FacesContext context) throws IOException {
20         final ResponseWriter writer = context.getResponseWriter();
21         writer.startElement("div", this);
22         writer.writeAttribute("style", "color : red", null);
23         writer.writeText("Hola adictosaltrabajo, hoy es: " + new
   java.util.Date(), null);
24         writer.endElement("div");
25     }
26
27 }
```

El nombre de la clase empieza por Html por convención, ya que el componente esta hecho para visualizarse en html por defecto. En el método encodeAll tenemos el código para que el componente se visualice en el cliente. La creacíon del html se realiza con un ResponseWriter que podemos recoger del contexto. Esta clase tiene varios métodos para escribir lenguaje de marcado entre ellos los que estamos utilizando aquí.

El método getFamily() devuelve la categoría general a la que pertenece el componente. Esto se utiliza para combinarlo con el rendererType para elegir un Renderer apropiado. En nuestro caso al no crear una familia de componentes ni de renderers podemos devolver null.

Y por último la anotacion @FacesComponent, que sería una alternativa a escribir el siguiente código en el faces-config.xml:

```
1 <component>
2     <component-type>HtmlCustomComponent</component-type>
3     <component-
  class>com.autentia.jsf.showcase.component.HtmlCustomComponent</component-class>
  </component>
```

Lo que hacemos con esto es registrar el componente para que en tiempo de ejecución se pueda construir.

Por último vamos a crear una tag library de Facelet para poder utilizar el componente en una vista.

```
01 <?xml version='1.0' encoding='UTF-8'?>
02 <facelet-taglib xmlns="http://java.sun.com/xml/ns/javaee"
03     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com
   /xml/ns/javaee/web-facelettaglibrary_2_0.xsd"
05     version="2.0">
06     <namespace>http://autentia.com/tutoriales</namespace>
07     <tag>
08         <tag-name>customComponent</tag-name>
09         <component>
10             <component-type>HtmlCustomComponent</component-type>
11         </component>
12     </tag>
13 </facelet-taglib>
```

Colocamos el taglib en WEB-INF/classes/META-INF/ para que la aplicación sea capaz de encontrarlo en tiempo de ejecución. Para que sea capaz de encontrarlo, el archivo debe acabar en .taglib.xml, en nuestro caso, custom.taglib.xml

Para usar el componente basta con crear un xhtml:

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
02         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml"
04     xmlns:h="http://java.sun.com/jsf/html"
05     xmlns:aut="http://autentia.com/tutoriales">
06
07 <h:head>
08     <title>Componente personalizado JSF 2.</title>
09 </h:head>
10
11 <h:body>
12     <aut:customComponent />
13 </h:body>
14 </html>
```

La URI del taglib debe ser la misma que hemos puesto en nuestro fichero xml, y como nuestro componente no necesita ningún atributo, con poner <aut:customComponent /> nos bastaría.

4. Añadir un parámetro al componente.

Una vez que ya tenemos nuestro componente creado, podría ser interesante que tuviera un parámetro de entrada para poder recibir el texto que queremos visualizar.

Cualquier atributo que se añada en la vista es accesible en nuestra clase java a través del mapa que devuelve getAttributes(). Vamos a añadir un atributo custommsg en nuestro .xhtml y cambiaremos el metodo encodeAll() para que escriba lo que haya en ese atributo.

```
...
@Override
    public void encodeAll(FacesContext context) throws IOException {
        final ResponseWriter writer = context.getResponseWriter();
        writer.startElement("div", this);
        writer.writeAttribute("style", "color : red", null);
        final String message = (String) this.getAttributes().get("custommsg");
        if (null == message) {
            writer.writeText("Hola adictosaltrabajo, hoy es: "
                    + new java.util.Date(), null);
        } else {
            writer.writeText(message, null);
        }
        writer.endElement("div");
    }
...
```

Ahora podemos utilizar <aut:customComponent custommsg="Mensaje pasado como parametro en el xhtml" /> y veremos este mensaje en el navegador.

Facelets soporta automáticamente EL, así que si escribimos como atributo <aut:customComponent custommsg="#{param.msg}" />. Con esto podríamos pasar el parámetro msg en la URL y se visualizaría en la página. Por ejemplo si accedemos a la url añadiendole: ?msg=Hola%20desde%20la%20url veremos "Hola desde la url" en nuestro navegador.

5. Referencias.

- http://jsfcompref.appspot.com/

6. Conclusiones.

Con este tutorial hemos podido ver lo básico para crear un componente en JSF 2. Algo que puede resultarnos útil si tenemos la necesidad de usar un componente que no existe en una librería de componentes.

Espero que os haya resultado útil. Cualquier duda, comentario o sugerencia podéis hacerla a continuación.

Un saludo.

César López.

COMENTARIOS