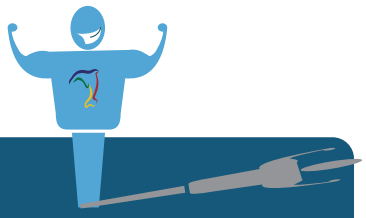
autentia

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...

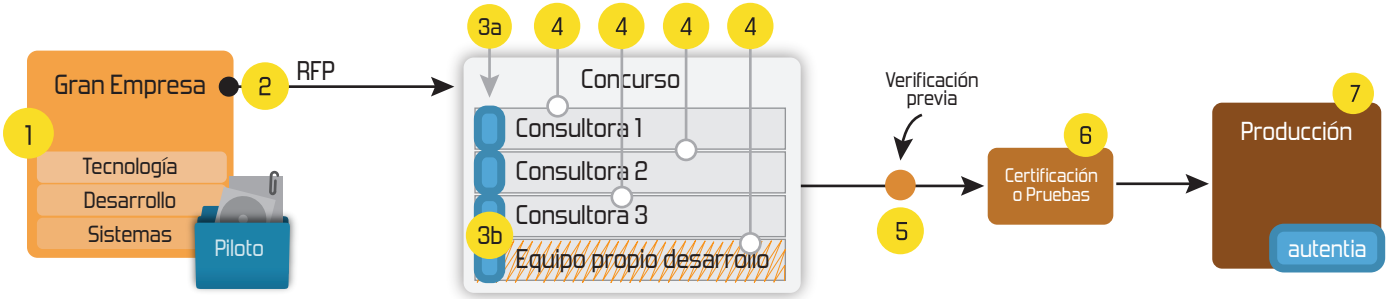
1. Desarrollo de componentes y proyectos a medida



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

DESARROLLADO POR:
Alberto Barranco Ramón

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: 2009-02-26

Cómo crear Widgets por composición en wuija

0. Índice de contenidos.

1. Introducción

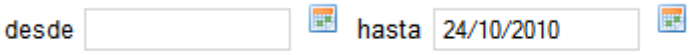
En este pequeño tutorial se pretende clarificar un poco cómo se crearían nuevos componentes visuales para wuija utilizando los ya existentes.

Pongamos que queremos crear un componente que permita insertar un rango de fechas, es decir una fecha "desde" y una fecha "hasta". En wuija tenemos ya un componente fecha que se llama DateProperty.

Sería el de la siguiente imagen:

Nuestro nuevo componente se va a llamar DateBetween, y va a ser una composición de dos DateProperty y dos OutputText, para que salgan el "hasta" y el "desde".

Debería quedar de la siguiente forma:



2. Creando la vista principal

Nuestro componente se va a mostrar en un formulario con otros campos, otros widgets, y otros componentes de Icefaces. Lo que vamos a hacer es montar nuestra vista y agregar nuestro widget DateBetween.

view plain copy to clipboard print ?

```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
03.     xmlns:ui="http://java.sun.com/jsf/facelets"
04.     xmlns:h="http://java.sun.com/jsf/html"
05.     xmlns:f="http://java.sun.com/jsf/core"
06.     xmlns:c="http://java.sun.com/jstl/core"
07.     xmlns:ice="http://www.icesoft.com/icefaces/component"
08.     xmlns:tnt="http://www.autentia.com/frmwrk/component">
09.
10.         <ui:composition>
11.         <ice:form id="globalForm">
12.         ...
13.             <tnt:widget childWidget="#{globalForm.dateBetween}" />
14.         ...
```

A esta vista la vamos a llamar **globalForm.jspx** y le vamos a asociar un controlador **GlobalForm.java**. Si nos fijamos, la línea 13 es la que contiene nuestro componente de wuija. globalForm haría referencia a nuestro Controlador y dateBetween hace referencia al widget.

Lo que estamos diciendo hasta ahora es que nuestro controlador como mínimo va a tener dentro un Widget del tipo DateBetween, y unos getters/setters para acceder a él. Eso es lo que implica esa línea. Aún no sabemos cómo se va a pintar nuestro componente ni cómo está formado, pero vayamos por partes.

3. Creando el Widget DateBetween

Cómo ya hemos dicho este componente va a ser una composición de dos DateProperty. Pero, ¿qué es un DateProperty?. Un DateProperty es un Widget de wuija que nos permite pintar un campo para recoger una fecha, es decir, esto:  DateProperty hereda de Property.java, que a su vez hereda de JsfWidget.java.

Si miramos dentro de DateProperty al final vemos que lo que pinta es un *ice:selectInputDate*, es decir el componente de Icefaces para recoger fechas, sólo que además tenemos funcionabilidad extra. Todo Widget debe heredar de JsfWidget. Esto aporta funcionabilidad al Widget. No es el objetivo de este tutorial explicar esta clase, porque podemos perder el foco de lo que estamos haciendo. Basta con decir que al extender de esta clase estamos implementando métodos que nos pueden servir para por ejemplo:

- Poder vincular Roles específicos asociados al Widget
- Saber si el Widget está activo o no.
- Añadir un Listener
- *...ver la clase para más imformación...*

Además al extender de JsfWidget estamos obligados a implementar el método *getRendererPath*. Si lo juntamos todo nos tendría que quedar un Widget más o menos como lo siguiente:

view plain copy to clipboard print ?

```
01. public class DateBetween extends JsfWidget {
02.
03.     private Date dateInit;
04.
05.     private Date dateEnd;
06.
07.     private DateProperty dateInitProperty;
08.
```

Reunión Madrid Ágil 02-11-2010: DDD (Domain Driven Design)

DataTable con paginación en base de datos con Primefaces

Resolviendo el cubo de Rubik (II) - ayudas, ejemplo Wink, extensiones y encuesta

Configuración de jCaptcha en Struts

MySQL - Sensibilidad a mayúsculas/minúsculas de los nombres de las tablas

Últimos Tutoriales del Autor

Cómo subir tutoriales a Adictos

Ejemplo de uso con JSF 2.0, Primefaces e Hibernate

Síguenos a través de:

Últimas ofertas de empleo

2010-10-11
Comercial - Ventas - SEVILLA.

2010-08-30
Otras - Electricidad - BARCELONA.

2010-08-24
Otras Sin catalogar - LUGO.

2010-06-25
T. Información - Analista / Programador - BARCELONA.

```
09.     private DateProperty dateEndProperty;
10.
11.     public DateBetween() {
12.         setDateInitProperty(new DateProperty(this.getClass
    (), "dateInit", true));
13.         setDateEndProperty(new DateProperty(this.getClass
    (), "dateEnd", true));
14.         this.dateEnd = new Date();
15.     }
16.
17.     public Date getDateInit() {
18.         return dateInit;
19.     }
20.     public void setDateInit(Date dateInit) {
21.         this.dateInit = dateInit;
22.     }
23.     public Date getDateEnd() {
24.         return dateEnd;
25.     }
26.     public void setDateEnd(Date dateEnd) {
27.         this.dateEnd = dateEnd;
28.     }
29.     public void setDateInitProperty(DateProperty dateInitProperty) {
30.         this.dateInitProperty = dateInitProperty;
31.     }
32.
33.     public DateProperty getDateInitProperty() {
34.         return dateInitProperty;
35.     }
36.
37.     public void setDateEndProperty(DateProperty dateEndProperty) {
38.         this.dateEndProperty = dateEndProperty;
39.     }
40.
41.     public DateProperty getDateEndProperty() {
42.         return dateEndProperty;
43.     }
44.
45.     @Override
46.     public String getRendererPath() {
47.         return RENDERER_PATH + "inputDateBetween.jspx";
48.     }
49.
50. }
```

Si nos fijamos un poco (lineas 7 y 9) vemos que este componente está formado por dos DateProperty, que era lo que ya teníamos. A esto se le llama hacer un componente por composición. Me creo uno nuevo a partir de otros.

Como vemos nuestra clase extiende de JsfWidget, por lo tanto implementa (linea 45-48) getRendererPath(). Aquí es donde le estamos diciendo cómo se va a pintar este componente. Los DateProperty (dateInitProperty y dateEndProperty) también tendrán su propio getRendererPath pues también son Widgets, y si navegáis hacia ellos veréis cómo lo que se está pintando al final es un *ice:selectInputDate*.

Como véis también tenemos dos fechas de tipo Date. Es aquí donde vamos a guardar los valores que el usuario introduzca por pantalla. En el constructor del Widget vemos que lo que estamos haciendo es inicializar cada uno de los Widget de los que éste está compuesto. De nuevo, no se van a explicar cómo están implementados los constructores específicos de los DateProperty, pues son componentes que teníamos de antes y el objetivo de este tutorial es explicar cómo se podrían montar nuevos componentes. Lo que sí tenemos que hacer es un constructor para nuestro nuevo componente (línea 11) en donde inicializamos los Widgest que lo componen y además por ejemplo ponemos la fecha "hasta" a día de hoy (línea 14).

También deberemos implementar los getters y setters para acceder a toda la información contenida en este widget (los dos dateProperty y las dos fechas).

3.1 ¿Cómo se pinta el nuevo Widget?

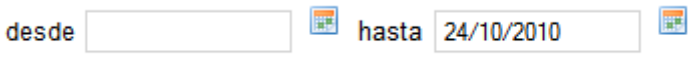
Precisamente la clase JsfWidget obliga a implementar getRendererPath() para indicarle al widget cómo debe pintarse. Si nos fijamos, nuestro nuevo Widget contenía en esa función *inputDateBetween.jspx*. Vamos a ver qué contendría esta vista.

```
view plain  copy to clipboard  print  ?
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!--

    Copyright 2008 Autentia Real Business Solutions S.L.

    This file is part of Autentia WUIJA.

    Autentia WUIJA is free software: you can redistribute it and/or modify

    it under the terms of the GNU Lesser General Public License as published b

    the Free Software Foundation, version 3 of the License.

    Autentia WUIJA is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License

    along with Autentia WUIJA. If not, see <http://www.gnu.org/licenses/>.


-->

<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" xmlns:ui="http://java.sun.c
    xmlns:h="http://java.sun.com/jsf/html" xmlns:f="http://java.sun.com/jsf/co
    xmlns:ice="http://www.icesoft.com/icefaces/component" xmlns:tnt="http://ww

    <ui:component>
        <ice:panelGrid columns="2">
            <h:panelGroup>
                <ice:outputText value="#{msg['inputDateBetween.from']}" />
                <tnt:widget childWidget="#{widget.dateInitProperty}">
                    <ui:param name="value" value="#{widget}" />
                </tnt:widget>
            </h:panelGroup>
            <h:panelGroup>
                <ice:outputText value="#{msg['inputDateBetween.to']}" />
                <tnt:widget childWidget="#{widget.dateEndProperty}">
                    <ui:param name="value" value="#{widget}" />
                </tnt:widget>
            </h:panelGroup>
        </ice:panelGrid>
    </ui:component>
</jsp:root>
```

Si nos fijamos a partir de la línea 27 lo que tenemos es un panel de 2 columnas. En la primera columna hemos agrupado el "desde" (ice:outputText value="#{msg['inputDateBetween.from']}") y un DateProperty (tnt:widget childWidget="#{widget.dateInitProperty}"). Este DateProperty es un widget, luego lo tenemos que declarar así. Widget está haciendo referencia a nuestro Widget DateBetween, que si os fijáis contiene un atributo dateInitProperty, que es un DateProperty, al cual podrá acceder porque tiene getter y setter.

Para el hasta lo haríamos igual. Con esto tendríamos hecha la vista y habríamos terminado nuestro Widget.



Espero que os haya sido de utilidad !!

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

Puedes opinar o comentar cualquier sugerencia que quieras comunicarnos sobre este tutorial; con tu ayuda, podemos ofrecerte un mejor servicio.

Enviar comentario
(Sólo para usuarios registrados)

» **Registrate** y accede a esta y otras ventajas «

COMENTARIOS