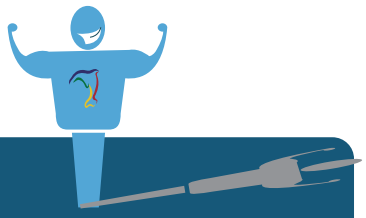
autentia

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...

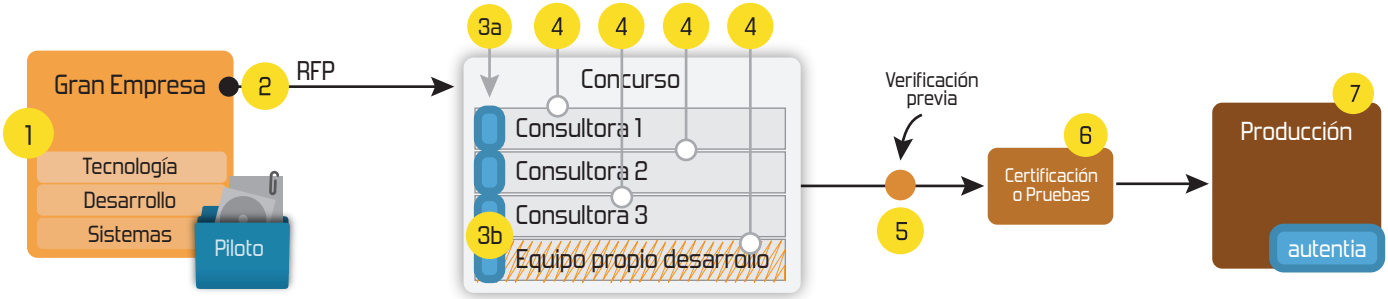
1. Desarrollo de componentes y proyectos a medida



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:
Jose Manuel Sánchez Suárez

Consultor tecnológico de desarrollo de proyectos informáticos.

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

Somos expertos en Java/J2EE

Fecha de publicación del tutorial: 2009-02-26

Gestión del evento de cambio de valor en JSF2: valueChangeListener.

0. Índice de contenidos.

1. Introducción

En este tutorial vamos a exponer la mejor manera de gestionar un evento de cambio de valor en la vista con JSF2.

Para ello haremos uso de un formulario de entrada de datos en el que los posibles valores de un componente dependen del valor que introduzca el usuario en otro. Se trata del típico ejemplo del selector de países que recarga el contenido del selector de ciudades.

Es requisito que la recarga del componente se produzca a través de Ajax, así iremos viendo que los cambios producidos en la vista repercuten en el managedBean puesto que tendremos un área en el que visualizan los mismos datos de entrada como datos de salida.

En el formulario habrá algún componente adicional, marcado como obligatorio para comprobar que:

- la petición que se lanza vía ajax no pierde los valores de los componentes visuales que se han rellenado hasta ese momento, y que
- cuando se realiza la selección del componente no salta el mensaje de obligatoriedad de otros campos, ese solo se produce al pulsar el botón de guardar.

2. Entorno.

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil MacBook Pro 17' (2.93 GHz Intel Core 2 Duo, 4GB DDR3 SDRAM).
- Sistema Operativo: Mac OS X Snow Leopard 10.6.1
- JSF2 (Mojarra 2.0.4)
- Tomcat 7.0.6

3. El ejemplo práctico.

Primero vamos a exponer la vista, el código fuente del árbol de componentes jsf. Estamos en un formulario de edición de un cliente, en el que tenemos una serie de campos:

- el nombre del cliente, que es obligatorio,
- una marca que indica si el cliente es vip,
- un selector de países, que al modificar el valor asignado provoca una recarga del siguiente selector,
- un selector de ciudades, cuyos posibles valores dependen del valor elegido en el selector de países.

```
01 <h:form id="pForm">
02   <h:panelGroup>
03     <h:panelGrid columns="3">
04
05       <h:outputLabel for="name" value="#{msg['Client.name']}"/>
06       <h:inputText id="name" value="#{clientView.client.name}" required="true">
07         <f:ajax render="clientOnServerSide" execute="@this" />
08       </h:inputText>
09       <h:message for="name" />
10
11       <h:outputLabel for="vip" value="#{msg['Client.vip']}"/>
12       <h:selectBooleanCheckbox id="vip" value="#{clientView.client.vip}" valueChangeListener="#
{clientView.vipChangeListener}"
13         immediate="true" >
14         <f:ajax render="clientOnServerSide" execute="@this" />
15       </h:selectBooleanCheckbox>
16       <h:message for="vip" />
17
18       <h:outputLabel for="country" value="#{msg['Client.country']}"/>
19       <h:selectOneMenu id="country" value="#{clientView.country}" converter="selectItemsConverter"
20         valueChangeListener="#{clientView.countryChangeListener}" immediate="true">
21         <f:ajax render="clientOnServerSide city" />
22         <f:selectItem itemLabel="#{msg['noSelectOption']}" noSelectionOption="true"/>
23         <f:selectItems value="#{clientView.countries}" var="country" itemValue="#{country}" itemLabel="#
{country.name}" />
24       </h:selectOneMenu>
25       <h:message for="country" />
26
27       <h:outputLabel for="city" value="#{msg['Client.city']}"/>
28       <h:selectOneMenu id="city" value="#{clientView.client.city}" converter="selectItemsConverter">
29         <f:ajax render="clientOnServerSide" execute="@this" />
30         <f:selectItems value="#{clientView.cities}" var="city" itemValue="#{city}" itemLabel="#{city.name}" />
31       </h:selectOneMenu>
32       <h:message for="city" />
33
34     </h:panelGrid>
35
36   </h:panelGroup>
37
38   <br />
39   <h:commandButton action="#{clientView.save}" value="#{msg['btn.save']}" />
```

```
  <h:messages />
  <hr />

  <h:panelGroup id="clientOnServerSide">
    <h:panelGrid columns="2">

      <h:outputText value="#{msg['Client.name']}" />
      <h:outputText value="#{clientView.client.name}" />

      <h:outputText value="#{msg['Client.vip']}" />
      <h:outputText value="#{clientView.client.vip}" />

      <h:outputText value="#{msg['Client.country']}" />
      <h:outputText value="#{clientView.country.name}" />

      <h:outputText value="#{msg['Client.city']}" />
      <h:outputText value="#{clientView.client.city.name}" />

    </h:panelGrid>
  </h:panelGroup>

</h:form>
```

Vamos a analizar los puntos más interesantes:

- las **líneas 5 a 9** contienen la etiqueta, el campo de entrada y los posibles mensajes relacionados con el **nombre** del cliente. Está marcado como obligatorio mediante el atributo required y tiene un evento ajax que al submitirse envía únicamente el componente de entrada y rerenderiza un panel inferior con los datos asignados al cliente en el managedbean.
- las **líneas 11 a 16** contienen la etiqueta, la casilla de verificación y los posibles mensajes relacionados con la marca de **"very important people"**, el check tiene un evento de cambio de valor que invoca a un escuchador del lado del managedBean y está marcado como immediate para que al producirse el evento e invoque al managedBean no provoque la validación del resto de campos que no están marcados con immediate.
- las **líneas 18 a 23** contienen la etiqueta, el selector y los posibles mensajes relacionados con el **país**, el selector tiene un evento de cambio de valor que invoca a un escuchador del lado del managedBean y, al igual que el anterior, está marcado como immediate. Además:
  - el evento ajax que lanza rerenderiza el área inferior con los campos de salida y, además, el componente siguiente que permite seleccionar la ciudad,
  - tiene un conversor que permite convertir de bean a selecItem y viceversa, para más detalles consultar el tutorial JSF2 return,
  - hace uso de la etiqueta f:SelectItem para establecer un valor por defecto vació de "Seleccione..."
  - hace uso de la etiqueta f:SelectItems para establecer los posibles países recuperados de un método del managedBean.
- las **líneas 27 a 32** contienen la etiqueta, el selector y los posibles mensajes relacionados con la **ciudad**, cuyos posibles valores depende del valor del campo anterior.
- las **líneas 43 y siguientes** contienen los campos de entrada convertidos en campos de salida.

El valor de todos los componentes del formulario, menos el país, se encuentran asociados a los correspondientes atributos de la entidad client del managedBean. El país no se almacena en el cliente, puesto que al almacenarse la ciudad, el país se deduce y evitamos redundancias.

El soporte necesario a la vista del lado del managedBean es el siguiente:

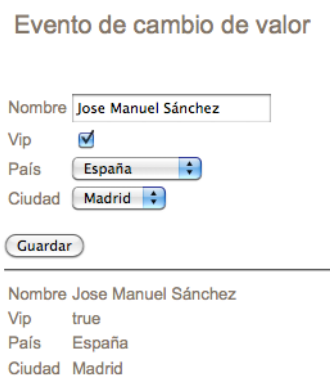
```
package com.autentia.jsf.showcase.view;

import java.io.Serializable;
import java.util.List;

import javax.annotation.PostConstruct;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
import javax.faces.context.FacesContext;
import javax.faces.event.ValueChangeEvent;

import org.apache.log4j.Logger;

import com.autentia.jsf.showcase.core.AddSampleData;
import com.autentia.jsf.showcase.core.HumanResourcesService;
import com.autentia.jsf.showcase.core.MetadataService;
import com.autentia.jsf.showcase.core.entities.Client;
import com.autentia.jsf.showcase.core.metadata.City;
import com.autentia.jsf.showcase.core.metadata.Country;

@ManagedBean
@ViewScoped
public class ClientView implements Serializable{

    private static final long serialVersionUID = -237761276054657507 8L;

    private static final Logger log = Logger.getLogger(ClientView.class.getName());

    private Client client;

    private Country country;

    @PostConstruct
    protected void init(){
        log.debug("init...");
        AddSampleData.initializeMetadata();
        client = new Client("");
    }

    public void setClient(Client client) {
        log.debug("setClient:" + client);
        this.client = client;
    }

    public Client getClient() {
        log.debug("getClient:" + client);
        return client;
    }

    public void setCountry(Country country) {
        log.debug("setCountry:" + country);
        this.country = country;
    }

    public Country getCountry() {
        log.debug("getCountry:" + country);
        return country;
    }

    public void save(){
        log.debug("saving client:" + client);
        try{
            HumanResourcesService.getInstance().addClient(client);
            FacesContext.getCurrentInstance().addMessage("", new FacesMessage("Información almacenada
correctamente."));
        }
        catch(Exception e){
            log.error("Exception saving the client",e);
            FacesContext.getCurrentInstance().addMessage("", new FacesMessage(e.getMessage()));
        }
    }

    public List<Country> getCountries(){
        log.debug("getCountries");
        return MetadataService.getInstance().getAllCountries();
    }

    public List<City> getCities(){
        log.debug("getCities");
        return MetadataService.getInstance().getCityByCountry(country);
    }

    public void vipChangeListener(ValueChangeEvent event){
        log.debug("vipChangeListener");
        // do nothing
    }
```

```
87
88      public void countryChangeListener(ValueChangeEvent event){
89          log.debug("countryChangeListener");
90          country = ( (Country) event.getNewValue());
91          City city = null;
92          if (!getCities().isEmpty()){
93              city = getCities().get(0);
94          }
95          client.setCity(city);
96          FacesContext.getCurrentInstance().renderResponse();
97      }
98
99  }
```

Nos vamos a centrar en los eventos que capturan los cambios de valor:

- el método **vipChangeListener** escucha el evento de cambio de valor del selector que marca al cliente como vip y no necesita tener ninguna lógica de control puesto que, de dicho componente no depende ningún otro y porque la lógica de rerenderización ya ha sido configurada en el componente f:ajax anidado en la vista,
- el método **countryChangeListener** sí tiene lógica de control porque queremos que al seleccionar un país se seleccione una ciudad por defecto, si no tuviéramos esa necesidad la lógica de control de este método también podría desaparecer. No siendo así, en este método:
  - se asigna a la variable country el valor del selector representado por el newValue,
  - se obtiene la primera ciudad del país, si la tuviera, para asignarla nivel de cliente.
  - se escapa del resto de fases del ciclo de vida de jsf (se ejecutará en la fase de APPLY_REQUEST_VALUES) para que no salten las validaciones ni se asignen los valores del resto de componentes al modelo.

El restulado a nivel de interaz de usuario sería el siguiente:



4. Identificación de posibles problemas.

Qué pasaría...

- si los componentes que tienen los eventos de cambio de valor no estuvieran marcados como immediate, no se llegaría nunca a la fase de UPDATE_MODEL_VALUES porque saltarían las validaciones del resto de campos marcados como obligatorios (o con algún error de validación) y no marcados como immediate.
- si el selector de países no tiene un valor por defecto y no inicializamos el valor del atributo correspondiente en el controlador, cuando se produce la primera submisión del formulario al managedBean se ejecuta el listener de cambio de valor, como éste tiene un renderResponse() vuelve a la vista sin producirse el evento que provocó la submisión. Si ponemos el ejemplo del botón de guardar, al pulsar no guardaría, solo asigna el valor del selector de ciudades, ya que se produce un evento de cambio de valor del null al primer valor del selector. El efecto es que el botón de guardar solo funciona si le pulsas dos veces (la segunda ya tiene asignado un valor).
- si no realizamos un renderResponse(), en función de qué componentes estén viajando al servidor en la petición ajax, puede que la lógica de control que modifica el valor de otro componente no tenga efecto porque se vea sobrescrito por el valor del componente que viaja en la request,
- si, por mis requisitos, las peticiones no pueden ser ajax, bastaría con eliminar los f:ajax e incluir en los componentes que tienen un evento de cambio de valor el atributo onchange="submit();".

5. Conclusiones.

Es batante más sencillo e incurrimos en un menor número de posibles errores trabajando con eventos de cambio de valor en JSF2 que en JSF1.2, en gran parte por el mayor nivel de granularidad que nos proporciona la rerenderización de componentes del atributo f:ajax, que se parece bastante a la que nos proporcionaba, hasta ahora, RichFaces a través de ajax4JSF.

Para aquellos que piensen que la gestión de ajax en JSF2 implica tener un mayor nivel de control en la vista sobre qué se repinta y que se envía en cada evento de cada componente, efectivamente y sí, pero yo lo prefiero y próximamente publicaremos un tutorial al respecto.

Un saludo.

Jose

jmsanchez@autentia.com

COMENTARIOS