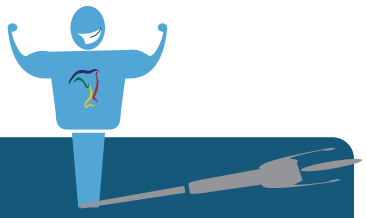
autentia

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...

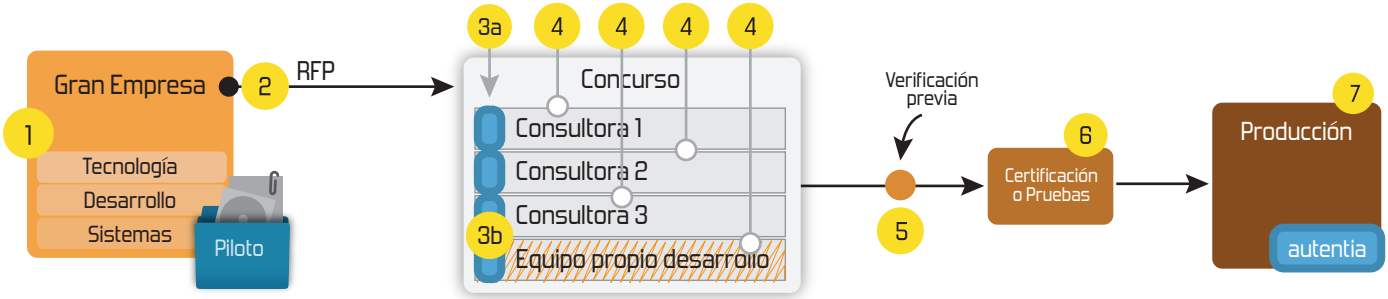
1. Desarrollo de componentes y proyectos a medida



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

Francisco Ferri Pérez

Consultor tecnológico de desarrollo de proyectos informáticos.

Desarrollador de proyectos informáticos, Microsoft Certified IT Professional - Enterprise Administrator

Ver todos los tutoriales del autor

Fecha de publicación del tutorial: 2013-01-29 Tutorial visitado 1 veces Descargar en PDF

Framework ZK

Crear el interfaz de usuario programando directamente desde un Richlet.

Enero de 2013, artículo original de Hawk Chen, Engineer, Potix Corporation

Contenido

Introducción

Un Richlet es un pequeño programa en Java que construye el interfaz de usuario mediante programación en Java, y esto es una alternativa a escribir ficheros ZUL que se sirvan bajo petición del usuario.

Cuando un usuario hace un apetición mediante una URL (request) que está mapeada en el Richletf, ZK Loader hace responsable al Richlet de la construcción de la interfaz de usuario.

Esto les da a los desarrolladores todo el poder a la hora de crear los componentes. La elección entre páginas ZUML y Richlets depende de su preferencia.

En este artículo, demostraremos cómo usar un Richlet creando una pequeña aplicación que consistirá en un buscador. Para una explicacíon más detallada ver: Richlet en la ZK Developer's Reference.

Aplicación de ejemplo

La aplicación de ejemplo que vamos a construir es un simple catálogo de coches.

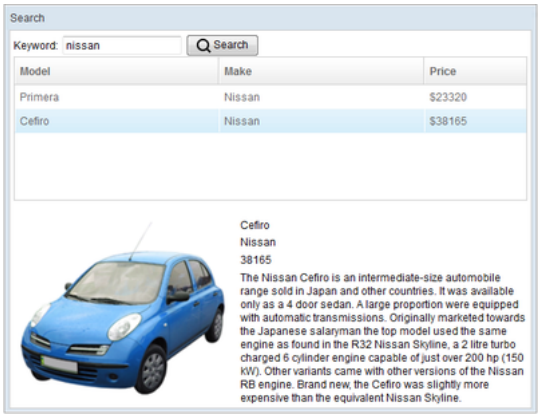
Esta aplicación tiene 2 funciones principales

1. Buscar coches.

   Escribir una palabra clave en el campo de texto, hacer click en el botón de buscar y mostrar los resultados de la búsqueda en la lista contigua.

2. Ver los detalles de un coche.

   Haciendo clic en uno de los coches de la lista, el área contigua mostrará los detalles del coche seleccionado, incluyendo el modelo, precio, descripción y vistra prévia.



Implementación

Para implementar un Richlet, tu clase tiene que implementar el interfaz Richelt. A pesar de que, no es necesario implementarla tal cual. En vez de eso puedes extender GenericRichlet y solo sobreescribir el método Richlet.service(Page). Este método será el que se llame cuando se haga una petición al Richlet por parte del usuario.

```
view plain print ?
01. public class SearchRichlet extends GenericRichlet {
02.
03.     @Override
04.     public void service(Page page) throws Exception {
05.         //...
06.     }
07.
08.     @Override
09.     public void init(RichletConfig config) {
10.         //initialize resources, e.g. get initial parameters
11.     }
12.
13.     @Override
14.     public void destroy() {
15.         //destroy resources
16.     }
17. }
```

Para tener mejor control, puedes incluso implementar los métodos Richlet.init(RichletConfig) y Richlet.destroy() para inicializar y destruir cualquier cosa que necesite el Richlet para funcionar.

Creando la Interfaz de Usuario

Una de las funciones princpales de un Richlet es crear la interfaz de usuario. Es recomendable empezar leyendo los conceptos básicos sobre intefaz de usuario en ZK antes de crearla mediante Java.

- El interfaz de usuario en ZK es un árbol de componentes, cada componente tiene un componente padre, y a su vez puede tener o no múltiples componentes hijos.
- No todos los componentes aceptan como hijos a todos los tipos de componentes que existen, solo a aquellos que se correspondan, o incluso ninguno.

Por ejemplo, un Grid en ZUL acpeta columnas y Filas como hijos. Para más detalle mirar la documentación al respecto: ZK Component Reference

```
view plain print ?
01. public class SearchRichlet extends GenericRichlet {
02.
03.     @Override
04.     public void service(Page page) throws Exception {
05.         Component rootComponent = buildUserInterface();
06.     }
07.
08.     private Component buildUserInterface(){
09.
10.         //build search area
11.         final Textbox keywordBox = new Textbox();
12.         Button searchButton = new Button("Search");
13.         searchButton.setImage("/img/search.png");
14.
15.         Hbox searchArea = new Hbox();
16.         searchArea.setAlign("center");
17.         searchArea.appendChild(new Label("Keyword:"));
18.         searchArea.appendChild(keywordBox);
19.         searchArea.appendChild(searchButton);
20.
21.         ...
22.     }
23. ...
24. }
```

Línea 12: Para crear un componente, simplemente lo instanciamos.
Línea 14: Cada atributo de un componente tiene su correspondiente método setter que podemos usar para asignarle el valor correspondiente.
Línea 18: Para establecer una relación Padre-Hijo entre componentes, podemos usar el método **appendChild(Component)**. Esto enlaza el componente indicado en el método, al final de la lista de componentes hijos (que ya tuviera) del componente padre. Hay otros métodos similares para hacer esto como **setParent(Component), insertBefore(Component,Component)**. Puedes encontrar más información al respecto en la documentación

Renderizando el modelo de datos

Después de proveer el objeto Modelo al componente ListBox, normalmente necesitamos indicar cómo mostrar los datos que contiene. Necesitamos crear un objeto Render para ello.

Un Render es una clase Java en la que se indica cómo se va a mostrar cada dato del modelo. Lo creamos implementando la interfaz de tipo Render que corresponda a cada componente, para un Listbox usaremos ListitemRenderer. En la ZK Developer's Reference están la del resto de componentes.

Item renderer de un Listbox

```
view plain print ?
01. class CarRenderer implements ListitemRenderer<Car>{
02.
03.     @Override
04.     public void render(Listitem listitem, Car car, int index)
05.         throws Exception {
06.         listitem.appendChild(new Listcell(car.getModel()));
07.         listitem.appendChild(new Listcell(car.getMake()));
08.         Listcell priceCell = new Listcell();
09.         priceCell.appendChild(new Label("$"));
10.         priceCell.appendChild(
11.             new Label(car.getPrice().toString()));
12.         listitem.appendChild(priceCell);
13.     }
14. }
```

- En el método *render()*, nos encargamos de crear el interfaz del usuario que queremos añadiendo los componentes hijos al componente padre Listitem.

Despues de crear el Render necesitamos asignarlo al ListBox.

```
view plain print ?
01. carListbox.setItemRenderer(new CarRenderer());
```

Event Listener

En nuestra aplicacíon de ejemplo, un usuario puede hacer click en el botón de "Search" para realizar su búsqueda, tenemos por lo tanto que escuchar el evento "onClick". Podemos resolverlo invocando al método AbstractComponent.addEventListener(String, EventListener) del comonente. El primer parámetro es el nombre del evento, y el segundo es un objeto que implmente la interfaz EventListener. En un EventListener, puedes manipular los componentes para añadir tu lógica de la aplicación, como puede ser cambiar atributos de otros componentes, crear nuevos componentes, o eliminar uno existente. Puedes ver todas las características de los componentes en la ZK Component Reference, y ver qué atributos puedes necesitar usar.

```
view plain print ?
01. public class SearchRichlet extends GenericRichlet {
02.
03.  private Component buildUserInterface(){
04.
05.    searchButton.addEventListener(
06.         Events.ON_CLICK, new EventListener<Event>() {
07.
08.    //search
09.    @Override
10.    public void onEvent(Event event) throws Exception {
11.         String keyword = keywordBox.getValue();
12.         List<Car> result = carService.search(keyword);
13.         carListbox.setModel(
14.             new ListModelList<Car>(result));
15.    }
16.
17.    });
18.
19.   ...
20.  }
21. }
```

Línea 7: Puedes crear una clase a parte que implemente EventListener. En este ejemplo usamos una clase anónima por simplificar. Pero ojo, en un clúster debes implementar la versión serializable SerializableEventListener.
Línea 10: Escribe la lógica de tu aplicación dentro del método onEvent(), cosas como leer un valor con un get, cambiar un atributo de un componente o actualizar información

Otra función del ejemplo es que cuando el usuario selecciona un coche de la lista, mostramos sus detalles en el área de información. Para manejar esto, tenemos que escuchar el evento de seleccionar del componente Listbox, y entonces leer los atributos del coche y asignar los valores a los componentes correspondientes de la vista.

```
view plain print ?
01. public class SearchRichlet extends GenericRichlet {
02.
03.     private Component buildUserInterface(){
04.
05.         carListbox.addEventListener(Events.ON_SELECT,
06.             new EventListener<SelectEvent>() {
07.             //show selected item's detail
08.             @Override
09.             public void onEvent(SelectEvent event)
10.             throws Exception {
11.                 //get selection from listbox's model
12.                 Set<Car> selection =
13.                 ((Selectable<Car>)carListbox.getModel())
14.                 .getSelection();
15.                 if (selection!=null && !selection.isEmpty()){
16.                     Car selected = selection.iterator().next();
17.                     previewImage.setSrc(selected.getPreview());
18.                     modelLabel.setValue(selected.getModel());
19.                     makeLabel.setValue(selected.getMake());
20.                     priceLabel.setValue(
21.                     selected.getPrice().toString());
22.                     descriptionLabel
23.                     .setValue(selected.getDescription());
24.                 }
25.             }
26.         });
27.
28.         ...
29.     }
30. }
```

Configuración del Richlet

Hay 2 requisitos para que el Richlet esté disponible para la aplicación cliente.

1. Activar el soporte de Richlets (en WEB-INF/web.xml)
2. Mapear la URL (que puede ser estática o un patrón) a la que responderá el Richlet (in WEB-INF/zk.xml)

Activar el soporte de Richlets

Por defecto, los richlets están desactivados. Para activarlos, hay que añadir la siguiente declaracíon en el fichero WEB-INF/web.xml.

```
view plain print ?
<servlet-mapping>
    <servlet-name>zkLoader</servlet-name>
    <url-pattern>/zk/*</url-pattern>
</servlet-mapping>
```

De este ejemplo, puedes cambiar /zk/* a otro patrón (que corresponda con una URL válida) que tu quieras, como por ejemplo /do/*. Fíjate que no puedes mapear la expresión como si de una extensión de un fichero se tratase (como por ejemplo *.do) o se procesará como una página ZUML en vez de como un Richlet.

Mapear la URL a la que responderá el Richlet

Cada Richlet que implementes, debes declararlo en la configuración WEB-INF/zk.xml de la siguiente forma:

```
view plain print ?
<richlet>
    <richlet-name>SearchRichlet</richlet-name>
    <richlet-class>tutorial.richlet.SearchRichlet</richlet-class>
</richlet>

<richlet-mapping>
    <richlet-name>SearchRichlet</richlet-name>
    <url-pattern>/search</url-pattern>
</richlet-mapping>
```

Línea 6: Después de definir el Richlet, puedes mapearlo bajo tantas URLs como quieras mediante el elemento de configuración **richlet-mapping**

Puedes visitar http://localhost:8080/PROJECT_NAME/zk/search y ver tu Richlet. La URL especificada en el parámetro url-pattern debe empezar siempre con "/".

Si la URL termina en /*, se asocia a cualquier petición con el mismo prefijo. Para recibir la URL de la petición actual puedes chequear el valor que retorna el método getRequestPath de la página en la que te encuentres.

```
view plain print ?
public class MyRichlet extends GenericRichlet {

    @Override
    public void service(Page page) throws Exception {
        if ("/search/admin".equals(page.getRequestPath())){
            //build admin UI
        }else{
            //build normal UI
        }
        ...
    }
}
```

Sumario

ZK es tan versátil que provee de multiples opciones para que seas tú quien elija cómo construir la Interfaz de usuario, o incluso varias al mismo tiempo, dependiendo de tus preferencias o entorno.

Descargas y recursos interesantes

- El código fuente usado en el artículo puedes encontrarlo aquí
- Descargar ZK
- Artículo original de Hawk Chen

Este documento es un extracto de la documentación oficial del Framework ZK, traducido y ampliado por Francisco Ferri. Colaborador de Potix (creadores del Framework ZK). Si quieres contactar con él puedes hacerlo en franferri@gmail.com, en twitter @franciscoferri o en LinkedIn