

¿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

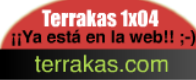
Tareas programadas (Quartz)
 Gestor documental (Alfresco)
 Inversión de control (Spring)

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

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





Soporte a desarrollo informático

Hosting patrocinado por **enredados**

Entra en Adictos a través de  

E-mail

Contraseña

Entrar [Deseo registrarme](#)
[Olvidé mi contraseña](#)

[Inicio](#) [Quiénes somos](#) [Formación](#) [Comparador de salarios](#) [Nuestro libro](#) [Más](#)

» Estás en: Inicio » Tutoriales » Tu aplicación web ZK enfoque MVC (4-5)



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: 2012-10-25

Tutorial visitado 1 veces [Descargar en PDF](#)

Framework ZK



Tu aplicación web ZK enfoque MVC (4-5)

Contenido

1. Introducción
2. Manejando la Lógica de la Interfaz de Usuario
3. Declarando los Controladores de la Interfaz de Usuario
4. Respondiendo a las Interacciones del Usuario
5. Controlando los Componentes de la Interfaz de Usuario
6. Mostrando un Conjunto de Datos
7. Implementando la Funcionalidad de "Ver Detalles"
8. Importar y Ejecutar la Aplicación de Ejemplo
9. Referencias

Introducción

Este tutorial pertenece a la siguiente serie:

1. [Configurar el entorno](#)
2. [Empezar a programar](#)
3. [MVC y MVVM ¿Qué son y en qué se diferencian?](#)
4. [MVC en ZK](#)
5. [MVVM en ZK](#)

Está dirigido a desarrolladores con experiencia en la creación de aplicaciones web en Java. A continuación aprenderás a manejar los componentes de la vista **manualmente** e implementar la lógica necesaria para que funcione nuestra aplicación.

A lo largo de este documento hacemos referencias a fuentes externas, las encontrarás todas en la sección de [Referencias](#)

Manejando la Lógica de la Interfaz de Usuario

El siguiente paso, después de crear el interfaz de usuario, es que esta, responda a la interacción del usuario. Lo que vamos a hacer a continuación es mostrarte **cómo controlar por tí mismo los componentes del interfaz de usuario**. Este podría estar considerado como el patrón de diseño **Modelo-Vista-Controlador (MVC)** [2].

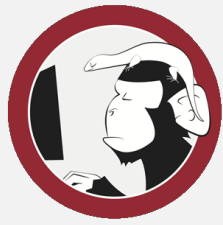
El patrón MVC divide la aplicación en 3 partes.

El modelo consta de los datos de la aplicación y las reglas de negocio. La clase **CarService** y otras clases usadas por ella representan la parte del modelo en nuestra aplicación de ejemplo.

La vista cubre la interfaz de usuario. La página zul que contiene los componentes de ZK representa esta parte. La interacción del usuario con los componentes de la vista genera eventos que son enviados a los controladores.

El controlador juega el rol de coordinador entre la Vista y el Modelo. En esencia, recibe eventos de la vista para actualizar el Modelo y recibe datos del Modelo para cambiar la presentación, es decir la Vista.

Catálogo de servicios Autentia



Síguenos a través de:



Últimas Noticias

» [Autentia estuvo en el Duatlón de Torrejón de Ardoz](#)

» [Participamos en la Carrera de las Empresas 2012](#)

» [¡¡¡Terrakas 1x04 recién salido del horno!!!](#)

» [Estreno Terrakas 1x04: "Terraka por un día"](#)

» [Nuevos cursos de gestión de la configuración en iOS y Android](#)

[Histórico de noticias](#)

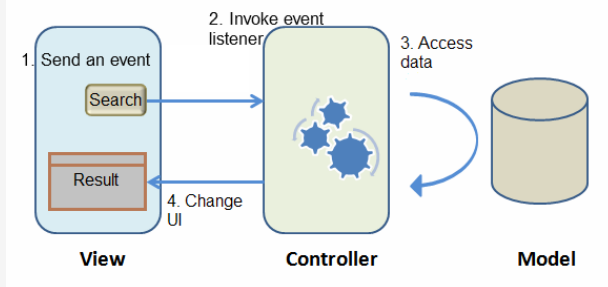
Últimos Tutoriales

» [Tu aplicación web ZK enfoque MVVM \(5-5\)](#)

» [Añadiendo reconocimiento de gestos a nuestra aplicación desde StoryBoard](#)

» [JRBeanCollectionDataSource: trabajando con arrays multidimensionales en Jasper Report.](#)

» [JRBeanCollectionDataSource: trabajando con colecciones de](#)



1. Cuando un usuario interactúa con un componente (por ejemplo hacemos clic en un botón) de la vista (archivo ZUL), esta interacción lanza un evento (onClick).
2. El evento es enviado al controlador e invoca el método correspondiente del "listener" (la clase que está escuchando a los eventos).
3. El método del "listener", normalmente, ejecuta la lógica de negocio o accede a los datos que necesita, para finalmente enviar las interacciones necesarias a los componentes de la vista.
4. Por lo tanto, cuando el estado de un componente cambia en el "listener", este cambio se propaga a la vista.

Declarando los Controladores de la Interfaz de Usuario

En ZK, el controlador es responsable de controlar los componentes de la vista y escuchar y lanzar los eventos que genera la interacción de los usuarios con los mismos. Para crear un controlador simplemente tenemos que extender la clase **org.zkoss.zk.SelectorComposer**:

```
package tutorial;

// omit import for brevity

public class SearchController extends SelectorComposer<Component> {

}
```

Después de crear el controlador, lo asociamos al componente de la vista correspondiente. Asociar un controlador con un componente requiere que indiquemos el nombre completo de la clase en el atributo **"apply"** en la página zul. A continuación vemos como asociar un controlador a un componente **"window"** en la vista.

Extraído de searchMvc.zul

```
<window title="Search" width="600px" border="normal"

apply="tutorial.SearchController">

<!-- omit other components for brevity -->

</window>
```

Puedes ver el fichero .zul completo en la sección de referencias en el siguiente link [\[3\]](#)

Después de asociar el controlador con el componente "window" de la vista, el controlador puede escuchar los eventos que se generan en la interfaz de usuario, tanto los del componente windows como los de los componentes hijos "child", y esto nos permite implementar las funciones de la aplicación.

Empecemos con la función de buscar: Un usuario escribe un texto, hace clic sobre el botón "Search" para lanzar la búsqueda.

Pasos para implementar la función necesaria:

1. Declaramos un método que escucha el evento, en este caso clic del botón.
2. Controlamos los componentes de la interfaz de usuario para implementar los cambios que requiera y ejecutamos la lógica de negocio en el método que recoge el evento.

Respondiendo a las Interacciones del Usuario

Cuando asociamos un controlador a un componente, todos los eventos lanzados por ese componente (y sus componentes hijos "child") son enviados al controlador. Si hay un método que hemos asignado para recoger el evento será invocado.

Por ejemplo si un usuario hace clic sobre el botón "Search", para lanzar la función de búsqueda, tenemos que escuchar el evento "onClick". Declaramos el método "search()", y lo asociamos al evento lanzado por el botón con la siguiente anotación:

```
@Listen(" [EVENT_NAME] = #[COMPONENT_ID] ")
```

Por lo tanto el método con la anotación queda como método "listener"

El código final quedaría como se muestra a continuación:

```
public class SearchController extends SelectorComposer<Component> {

    @Listen("onClick = #searchButton")
    public void search(){

    }

}
```

- Línea 3: "searchButton" es el id del componente en el fichero zul. Además, la anotación @Listen soporta más parámetros para encontrar el componente en el fichero zul. Más información en las referencias [\[4\]](#)
- Línea 4: Tiene que ser un método de ámbito público.

datos básicos en Jasper Report.

» Código de barras con iReport

Últimos Tutoriales del Autor

» Tu aplicación web ZK enfoque MVVM (5-5)

» MVC y MVVM (3-5)

» Empezar a programar con ZK (2-5)

» Tu primer proyecto web con ZK (1-5)

» NIC Bonding, NIC Teaming, Port Trunking, Etherchannel o Ether bonding, con ifenslave en Ubuntu

Categorías del Tutorial

» Otras Técnicas

Últimas ofertas de empleo

2011-09-08
Comercial - Ventas - MADRID.

2011-09-03
Comercial - Ventas - VALENCIA.

2011-08-19
Comercial - Compras - ALICANTE.

2011-07-12
Otras Sin catalogar - MADRID.

2011-07-06
Otras Sin catalogar - LUGO.

Controlando los Componentes de la Interfaz de Usuario

Después de establecer la relación entre el evento de la vista (zul) y el método del "Listener", podemos empezar a implementar la lógica de negocio. Pero primero tenemos que definir en el controlador, los componentes de los que escuchamos sus eventos mediante la anotación **@Wire**.

Pasos para recibir eventos de los componentes:

1. Declaramos una variable con el tipo del componente (por ejemplo Listbox, Label...)
2. La variable tiene que llamarse como el id (en la página zul) del componente que representa.
Podemos hacer esto de diferentes forma, esta es la forma por defecto para la anotación **@Wire**, si quieres saber otra forma de asociar componentes visita la Guía de Referencia para Desarrolladores de ZK [5] para descubrir otras formas.
3. Añadimos la anotación **@Wire** a la variable que hemos creado.

Entonces ZK "conecta" el componente de la vista ZUL a la variable declarada. Después de que esto esté hecho, podemos controlar y manipular el interfaz de usuario directamente manipulando las variables que hemos declarado.

```
public class SearchController extends SelectorComposer<Component> {

    @Wire
    private Textbox keywordBox;
    @Wire
    private Listbox carListbox;

    //other codes...
}
```

- Línea 5-6: en el fichero searchMVC.zul hay un listbox en el que su id es "**carListbox**". ZK creará la relación entre la variable "carListbox" y el componente de la vista de tipo listbox, después de que los componentes de la vista sean creados.

El método de la búsqueda contiene una lógica simple: Llama a la clase de servicio "carService" con el texto que buscamos e inserta los resultados en la lista "listbox". De una variable que está referenciada/unida a un componente, podemos conocer el valor de sus atributos a través de getters (por ejemplo getValue()) o cambiar el estado del componente visual, por ejemplo haciendo un componente de etiqueta "label" invisible mediante un setter (por ejemplo setVisible(false)) para conseguir una respuesta dinámica en el interfaz de usuario.

Por lo tanto, podemos obtener el texto escrito por el usuario mediante "keywordBox.getValue()" y cambiar la información de la lista "listbox" mediante "carListbox.setModel()". El modelo de un componente es la información que este componente contiene, y podemos cambiar el modelo cambiando los datos directamente en la pantalla.

```
public class SearchController extends SelectorComposer<Component> {
    //omit codes to get components

    @Listen("onClick = #searchButton")
    public void search(){
        String keyword = keywordBox.getValue();
        List<Car> result = carService.search(keyword);
        carListbox.setModel(new ListModelList<Car>(result));
    }
}
```

- Línea 8: Nótese que "setModel()" solo acepta un objeto de tipo "ListModel", por lo tanto podemos usar "**org.zkoss.zul.ListModelList**" para encapsular/envolver la lista de resultados de la búsqueda. También hay otros objetos de tipo "ListModel" para diferentes tipos de listas y colecciones, más información al respecto en la sección de referencias [6]

Mostrando un Conjunto de Datos

Hemos conseguido que se invoque la búsqueda correctamente al hacer clic, pero todavía queremos mostrar en el listbox los resultados de la búsqueda correctamente. Ese es el motivo por el cual no hemos especificado cómo mostrar el modelo de datos en el componente "listbox". Ahora usaremos una etiqueta de zul especial, "<template>" [7], para controlar lo que mostramos de cada resultado.

ZK mostrará la etiqueta "<template>" iterativamente para cada objeto del modelo, es decir de los resultados.

Pasos para usar "<template>":

1. Usar "<template>" para encapsular componentes que queremos repetir iterativamente.
2. Establecemos el atributo "name" del "<template>" como "modelo".[8]
3. Usamos la variable implícita del lenguaje zul "**each**" para asignar las propiedades del objeto de dominio "Car" a los correspondientes atributos del componente visual.

Extraído de searchMVC.zul

```
<listbox id="carListbox" height="160px" emptyMessage="No car found in the result">

<listhead>
<listheader label="Name" />
<listheader label="Company" />
<listheader label="Price" width="20%"/>
</listhead>

<template name="model">
<listitem>
<listcell label="{each.name}"></listcell>
<listcell label="{each.company}"></listcell>
<listcell><label value="{each.price}" /></listcell>
</listitem>
</template>
```

</listbox>

- Línea 7: La etiqueta "<template>" tiene que estar dentro del listBox.
- Línea 8: La etiqueta "<listitem>" del ejemplo inicial/anterior (estática), debemos reemplazarla por el código de este ejemplo.
- Línea 9: La variable "each" representa el objeto de dominio en el modelo, el cual es "Car" en esta aplicación de ejemplo. Podemos usarla para acceder a las propiedades del objeto de dominio mediante Expression Language (EL), por ejemplo "\${each.name}"

Implementando la Funcionalidad de "Ver Detalles"

Las secciones anteriores describen los pasos básicos para implementar una función en ZK. Veamos a continuación cómo aplicar todos los pasos aprendidos para implementar la funcionalidad para ver detalles de nuestra aplicación. Declaramos el método para escuchar el evento "onSelect" de la lista "listbox" con la anotación @Listen, entonces usaremos la anotación @Wire para escuchar componentes, incluyendo "previewImage", "nameLabel", "priceLabel" y "descriptionLabel" y les asignamos valores con los métodos setter.

```
public class SearchController extends SelectorComposer<Component> {

    @Wire
    private Listbox carListbox;
    @Wire
    private Label nameLabel;
    @Wire
    private Label companyLabel;
    @Wire
    private Label priceLabel;
    @Wire
    private Label descriptionLabel;
    @Wire
    private Image previewImage;

    @Listen("onSelect = #carListbox")
    public void showDetail(){
        Car selected = carListbox.getSelected().getValue();
        previewImage.setSrc(selected.getPreview());
        nameLabel.setValue(selected.getName());
        companyLabel.setValue(selected.getCompany());
        priceLabel.setValue(selected.getPrice().toString());
        descriptionLabel.setValue(selected.getDescription());
    }
    //omit other codes for brevity
}
```

Para obtener el código fuente completo del fichero .zul, por favor mira la sección de referencias [9]

Importar y Ejecutar la Aplicación de Ejemplo

A continuación puedes descargar el código fuente completo de la aplicación de ejemplo, preparado sobre un proyecto de Eclipse que puedes importar directamente en él, de forma que no tengas que empezar desde 0.

Para usar la aplicación de ejemplo sigue los siguientes pasos:

1. Descarga el zip que contiene la aplicación de ejemplo de [Aquí](#)
2. En Eclipse, selecciona **"File / Import / General / Existing Projects into Workspace"**, y elige **"Select archive file"** para importar el fichero .zip que contiene el ejemplo como proyecto en tu Eclipse.
3. Y sigue las instrucciones del tutorial de esta serie correspondiente a [Configurar el entorno](#) para arrancarlo.

Referencias

- [1] [Página web oficial de ZK](#)
- [2] [MVC, guía del desarrollador](#)
- [3] [searchMvc.zul](#)
- [4] [Sintaxis del selector](#)
- [5] [Enlazar componentes de la vista con el controlador](#)
- [6] [Tipos de listas](#)
- [7] [Template / Plantilla](#)
- [8] [Template / Plantilla para un componente ListBox](#)
- [9] [SearchController.java](#)

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 [Linkedln](#)

A continuación puedes evaluarlo:

[Regístrate para evaluarlo](#)



Por favor, vota +1 o compártelo si te pareció interesante



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

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



SOME RIGHTS RESERVED

Esta obra está licenciada bajo [licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5](#)

Copyright 2003-2012 © All Rights Reserved. [Legal](#) [Contacto](#) [Condiciones de uso](#) | [Banners](#) | Powered by

