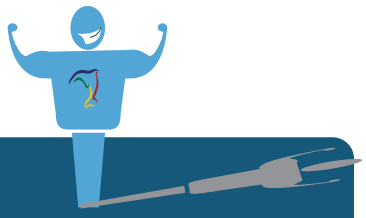
autentia

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...

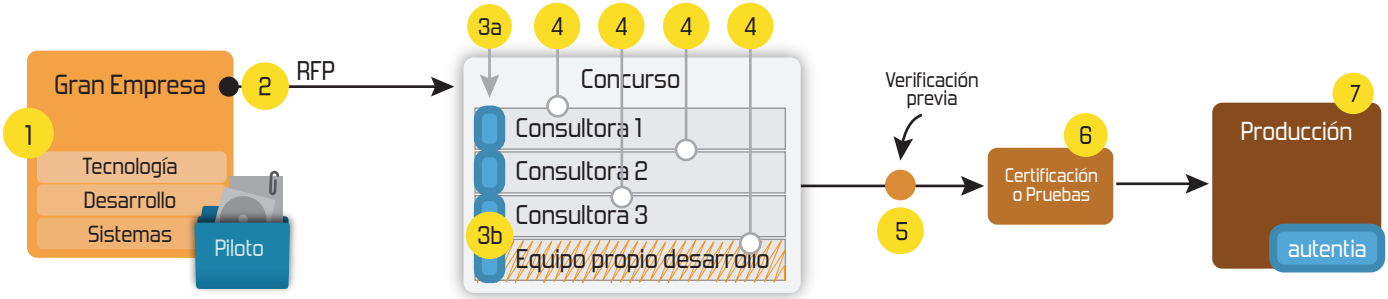
1. Desarrollo de componentes y proyectos a medida



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:
Rubén Aguilera Díaz-Heredero

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero en Informática, especialidad en Ingeniería del Software

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

Somos expertos en Java/J2EE

Fecha de publicación del tutorial: 2011-02-20

CRUD con Spring MVC Portlet (II)

0. Índice de contenidos.

1. Entorno

Este tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil Mac Book Pro 17" (2,6 Ghz Intel Core i7, 8 GB DDR3)
- Sistema Operativo: Mac OS X Snow Leopard 10.6.4
- Spring MVC Portlet 3.0.4
- Maven 2.2.1
- Eclipse 3.6 (Helios) con M2Eclipse
- Liferay 6.0.5
- displaytags 1.2

2. Introducción

Este tutorial es la segunda parte del tutorial CRUD con Spring MVC Portlet, por lo que se recomienda al lector que se lo lea antes de continuar con este.

En este tutorial lo primero que vamos a hacer es crear una vista donde se muestren las personas que fuimos dando de alta en el anterior tutorial. Para vamos a ver como la librería displaytags nos puede ayudar en la tediosa tarea de tener que crear una listado con paginación y ordenación de los campos.

Y como no es lo mismo contarlo que picarlo, voy a implementar y explicar esto, pero espero no tardar 21 días. ;)

3. Configurando nuestro proyecto para trabajar con displaytags

Para quien no lo conozca ya, solo decir que esta librería es la salvación cuando tenemos que enfrentarnos al típico problema de las tablas con paginación y ordenación, aunque es mucho más y toda la información la podéis encontrar en http://www.displaytag.org/1.2/

Para configurar nuestro proyecto (maravillas de trabajar con Maven) tenemos que añadir las siguientes dependencias a nuestro pom.xml:

```
view plain  print  ?
01. <dependency>
02.     <groupid>displaytag</groupid>
03.     <artifactid>displaytag</artifactid>
04.     <version>1.2</version>
05. </dependency>
06. <dependency>
07.     <groupid>displaytag</groupid>
08.     <artifactid>displaytag-portlet</artifactid>
09.     <version>1.2</version>
10. </dependency>
```

A fin de configurar el soporte de portlet de la librería tenemos que crear dentro de nuestra carpeta src/main/resources un fichero llamado displaytag.properties con el siguiente contenido:

```
view plain  print  ?
01. factory.requestHelper=org.displaytag.portlet.PortletRequestHelperFactory
```

4. Creando el listado de personas dadas de alta

Recordad que partimos del proyecto que creamos en el anterior tutorial de la serie y que incluso ya tenemos creada la vista a la que convenientemente llamamos listado_personas.jsp y que invocávamos desde la vista por defecto de nuestro portlet.

Lo primero que tenemos que hacer es editar este jsp, que recordad solo tenía hasta ahora un enlace al alta de la persona.

Aquí ponemos en juego la librería displaytags, para que nos muestre un listado simple de las personas, añadimos el siguiente código al jsp:

```
view plain  print  ?
01. <%@ taglib uri="http://java.sun.com/portlet" prefix="portlet"%>
02. <%@ taglib uri="http://displaytag.sf.net" prefix="display"%>
03. <%@ taglib uri="http://java.sun.com/jstl/core_rt" prefix="c" %>
04.
05. <portlet:defineObjects />
06.
07. <c:if test="${msgstatus != null}">
08. <p class="portlet-msg-success">${msgstatus}</p>
09. </c:if>
10.
11. <portlet:renderurl var="detallePersonaAltaURL">
12.     <portlet:param name="futuraAccion" value="crud.alta"></portlet:param>
13. </portlet:renderurl>
14. <p><a href="${detallePersonaAltaURL}"><spring:message code="form.anadirPersona"></sp
15.
16. <display:table name="listaPersonas"/>
17.
```

Con esta llamada la librería requiere un atributo en sesión llamado "listaPersonas" que contenga una lista de elementos de la clase Persona.

Para hacer esto editamos nuestro controlador a fin de recuperar el listado de personas de la base de datos e introducirlo en sesión:

```
view plain  print  ?
01. @RequestMapping
02. protected final String defaultView(Model model) {
03.     List<Persona> listaPersonas = personaDAO.getAll(Persona.class);
04.     model.addAttribute("listaPersonas", listaPersonas);
05.     return "listado_personas";
06. }
```

Esto va a generar una tabla en HTML con la información de las personas almacenadas, para darle un aspecto más vistoso, podemos crear el fichero .css que añada estilo a la tabla pero no hay que olvidar importarlo desde el fichero css/main.css que ya existe en nuestro proyecto de esta forma si creamos el fichero displaytags.css tendríamos que añadir la siguiente línea:

ZK - ¿Cómo crear tu primer proyecto con ZK?

CRUD con Spring MVC Portlet

REST y como hacer con jQuery un PUT hacia Spring MVC

Últimos Tutoriales del Autor

CRUD con Spring MVC Portlet

Librería de acceso a datos con Spring y JPA

Ejemplo de Swing Worker: ¿Por qué se me congela la interfaz?

Utilización de Commons Digester para un sistema de preferencias configurable

Ejemplo básico de Spring MVC Portlet

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.

```
view plain print ?
01. @import url("displaytag.css");
```

Y este podría ser el resultado:

Ahora vamos a personalizar la tabla añadiendo nombres a las columnas, los enlaces para las acciones de edición y borrado, la paginación y la ordenación en memoria.

Esto se consigue con el siguiente código:

```
view plain print ?
01. <display:table name="listaPersonas" id="persona" pagesize="2">
02.     <display:column property="nombre" titlekey="persona.nombre" sortable="true"></di
03.     <display:column property="apellidos" titlekey="persona.apellidos"></display:colu
04.     <display:column property="direccion" titlekey="persona.direccion"></display:colu
05.     <display:column>
06.         <portlet:renderurl var="detallePersonaModificacionURL">
07.             <portlet:param name="futuraAccion" value="crud.modificacion"></portlet:p
08.             <portlet:param name="id" value="${persona.idPersona}"></portlet:param>
09.         </portlet:renderurl>
10.         <a href="${detallePersonaModificacionURL}"><spring:message code="crud.modifi
11.     </spring:message></display:column>
12.     <display:column>
13.         <portlet:renderurl var="detallePersonaBajaURL">
14.             <portlet:param name="futuraAccion" value="crud.baja"></portlet:param>
15.             <portlet:param name="id" value="${persona.idPersona}"></portlet:param>
16.         </portlet:renderurl>
17.         <a href="${detallePersonaBajaURL}"><spring:message code="crud.baja">
18.     </spring:message></display:column>
19. </display:table>
```

Fijaos como hemos indicado la paginación de la tabla simplemente añadiendo el atributo pagesize al valor que queramos mostrar por página y la ordenación añadiendo el atributo sortable a true en aquellos campos por lo que queremos ordenar la tabla, en nuestro caso, solo queremos poder ordenar por el nombre. El resultado es el siguiente:

Los nombres de las columnas las tenemos que añadir al fichero messages.properties y para internacionalizar los mensajes propios de la librería displaytags como los que se utilizan en la paginación. Para ello añadimos las siguientes claves dentro del fichero displaytags.properties:

```
view plain  print  ?
01. basic.msg.empty_list=No hay registros
02. basic.empty.showtable=false
03. basic.msg.empty_list_row=No hay registros
04. sort.amount=list
05. paging.banner.placement=bottom
06. paging.banner.no_items_found=<span class="pagebanner">No {0} encontrados.</span><br>
07. paging.banner.one_item_found=<span class="pagebanner">1 {0} encontrado.</span><br>
08. paging.banner.all_items_found=<span class="pagebanner">{0} {1} encontrados, total {0
09. paging.banner.some_items_found=<span class="pagebanner">{0} {1} encontrados, mostran
10. paging.banner.full=<span class="pagelinks">[<a href="{1}">Primera</a>/<a href="
    {2}">Anterior</a>] {0} [<a href="{3}">Siguiente</a>/<a href="
    {4}">Última</a>]</span>
11. paging.banner.first=<span class="pagelinks">[Primera/Anterior] {0} [<a href="
    {3}">Siguiente</a>/<a href="{4}">Última</a>]</span>
12. paging.banner.last=<span class="pagelinks">[<a href="{1}">Primera</a>/<a href="
    {2}">Anterior</a>] {0} [Siguiente/Última]</span>
13. paging.banner.page.link=<a href="{1}" title="Ir a la página {0}"> {0}   </a>
```

En esta URL http://www.displaytag.org/1.2/configuration.html tenéis el listado completo de claves que se pueden internacionalizar.

5. Añadimos las acciones de edición y borrado

En este momento ya tenemos todo preparado para añadir la lógica de edición y borrado de la persona, de forma que si el usuario pulsa en una de estas acciones se muestre la pantalla de detalle de la persona para modificar los datos o borrar el registro.

Para ello vamos a añadir la lógica de redirección modificando el método redirectURL de nuestro controlador de esta forma:

```
view plain  print  ?
01. @RequestMapping(params = "futuraAccion")
02. protected final String redirectURL(Model model,
03.     @RequestParam("futuraAccion") String futuraAccion,
04.     @RequestParam(required = false, value = "id") Long id) {
05.     if ("crud.alta".equals(futuraAccion)) {
06.         model.addAttribute("personaForm", new PersonaForm());
07.     } else {
08.         final Persona persona = personaDAO.findByPK(Persona.class, id);
09.         model.addAttribute("personaForm", new PersonaForm(persona));
10.     }
11.     model.addAttribute("futuraAccion", futuraAccion);
12.     return "detalle_persona";
13. }
```

Esto rellenará los campos del formulario con los datos del registro seleccionado y preparará la siguiente acción que vamos a capturar añadiendo las siguientes funciones en el controlador:

```
view plain  print  ?
01. @RequestMapping(params = {"javax.portlet.action=ejecutarAccion", "accion=crud.modifi
02. protected final void modificarPersona(
03.         @ModelAttribute("personaForm") PersonaForm personaForm,
04.         @RequestParam(required=false, value="form.cancelar") String cancelar, Model
05.
06.     if (siUsuarioNoCancelaAccion(cancelar)) {
07.         personaDAO.update(personaForm.getPersona());
08.         model.addAttribute("msgstatus", messages.getMessage("status.ok", null, Local
09.     }
10. }
11.
12. @RequestMapping(params = {"javax.portlet.action=ejecutarAccion", "accion=crud.baja"}
13. protected final void bajaPersona(
14.         @ModelAttribute("personaForm") PersonaForm personaForm,
15.         @RequestParam(required=false, value="form.cancelar") String cancelar, Model
16.
17.     if (siUsuarioNoCancelaAccion(cancelar)) {
18.         personaDAO.remove(personaForm.getPersona());
19.         model.addAttribute("msgstatus", messages.getMessage("status.ok", null, Local
20.     }
21. }
```

6. Conclusiones

Como veis vamos añadiendo lógica a medida que la vamos necesitando tratando de introducir el mínimo código posible y no reinventando la rueda con la utilización de librerías tan útiles como

displaytags.

En el próximo tutorial vamos a ver cómo añadir validación al usuario y pruebas unitarias.

Cualquier duda o sugerencia en la zona de comentarios.

Saludos.

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