

¿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)

Estás en:

[Inicio](#) [Tutoriales](#) Utilización de Commons Digester para un sistema de preferencias configurabl...



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

[Catálogo de servicios Autentia](#)

[Anuncios Google](#)

[Java](#)

[Java Tutorial String](#)

[Programming in Java](#)

Fecha de publicación del tutorial: 2011-01-17



Share |

[Regístrate para votar](#)

Utilización de Commons Digester para un sistema de preferencias configurable

0. Índice de contenidos.

- [1. Entorno](#)
- [2. Introducción](#)
- [3. Manos a la obra](#)
- [4. Conclusiones](#)

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
- Maven 2.2.1
- Eclipse 3.6 (Helios) con M2Eclipse
- Commons Digester 2.1

2. Introducción

Hoy quiero hablar de una librería que siempre ha estado con nosotros pero que nunca la hemos utilizado directamente. Se trata de otra de las librerías de la maravillosa serie Commons de Apache, Commons Digester. Toda la información sobre esta librería la podéis encontrar en la URL: <http://commons.apache.org/digester/>

Nosotros vamos a utilizarla para crear un sistema de preferencias que sea configurable y extensible, es decir, que respetando la estructura de un XML podamos modificar los valores de las preferencias en función de un tipo, sin tener que tocar ni una línea de código.

3. Manos a la obra

Lo primero que tenemos que hacer es crear un proyecto con Maven e incluir la librería Commons Digester como dependencia en el pom.xml

Últimas Noticias

[Online Java Class](#)

 [XIV Charla Autentia - ZK](#)

 [Informática profesional: Las reglas no escritas para triunfar en la empresa. 2ª EDICIÓN ACTUALIZADA.](#)

 [Pequeño coding dojo con Carlos Ble en las oficinas de Autentia.](#)

 [Disponible gratis, Autentia Comic para el iPhone y iPad,](#)

 [Comentando #AID2010. Agil Industrial Day 30 Nov 2010](#)

 [Histórico de NOTICIAS](#)

Últimos Tutoriales

 [Introducción a Apache James](#)

 [MongoDB, primeros pasos](#)

 [Spring @Configurable y los modelos de dominio anémicos](#)

```

view plain print ?
01. <dependency>
02.   <groupid>commons-digester</groupid>
03.   <artifactid>commons-digester</artifactid>
04.   <version>2.1</version>
05. </dependency>

```

Lo siguiente es confeccionar el XML que va a almacenar las dependencias. Para nuestro ejemplo nos bastará con este:

```

view plain print ?
01. <preferences>
02.   <preference type="default">
03.     <datasource>
04.       <url>url-default</url>
05.       <driver>driver-default</driver>
06.       <user>user-default</user>
07.       <password>password-default</password>
08.     </datasource>
09.     <idioma>idioma-default</idioma>
10.   </preference>
11. </preferences>

```

Cada uno de los juegos de preferencia lo vamos a identificar con un atributo type, de modo que podamos añadir tantos juegos de preferencia como queramos.

Lo siguiente que tenemos que hacer es crear las clases en función del XML que tengamos. En mi caso podemos identificar las siguientes clases:

```

view plain print ?
01. package com.autentia.preferences;
02.
03. public class Datasource {
04.
05.     private String url;
06.     private String driver;
07.     private String user;
08.     private String password;
09.
10.     public String getUrl() {
11.         return url;
12.     }
13.     public void setUrl(String url) {
14.         this.url = url;
15.     }
16.     public String getDriver() {
17.         return driver;
18.     }
19.     public void setDriver(String driver) {
20.         this.driver = driver;
21.     }
22.     public String getUser() {
23.         return user;
24.     }
25.     public void setUser(String user) {
26.         this.user = user;
27.     }
28.     public String getPassword() {
29.         return password;
30.     }
31.     public void setPassword(String password) {
32.         this.password = password;
33.     }
34.
35.     public String toString(){
36.         return new StringBuilder("URL: ").append(url)
37.             .append(" driver: ").append(driver).append(" user: ")
38.             .append(user).append(" password: ").append(password).toString();
39.     }
40.
41. }

```

 [Madrid.rb y la kata de los Romanos](#)

 [Y un Jamón!](#)

Últimos Tutoriales del Autor

 [Ejemplo básico de Spring MVC Portlet](#)

 [Trabajando con los Web Services de Liferay](#)

 [Liferay IDE](#)

 [CAS: Validador personalizado](#)

 [CAS: Personalización de la interfaz](#)

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. package com.autentia.preferences;
02.
03. public class Preference {
04.
05.     private String type;
06.     private Datasource datasource;
07.     private String idioma;
08.
09.     public void setType(String type) {
10.         this.type = type;
11.     }
12.     public String getType() {
13.         return type;
14.     }
15.     public Datasource getDatasource() {
16.         return datasource;
17.     }
18.     public void setDatasource(Datasource datasource) {
19.         this.datasource = datasource;
20.     }
21.     public String getIdioma() {
22.         return idioma;
23.     }
24.     public void setIdioma(String idioma) {
25.         this.idioma = idioma;
26.     }
27.
28.     public String toString(){
29.         return new StringBuilder("Type: ").append(type)
30.             .append(" Datasource: ").append(datasource)
31.             .append(" Idioma: ").append(idioma).toString();
32.     }
33.
34. }

```

Estas dos clases se corresponden con las etiquetas del XML. Ahora tenemos que tener en cuenta que vamos a tener más de un juego de preferencias por lo que la clase Preferences.java tiene que tener un vector que las almacene. Por lo tanto también vamos a necesitar un método que almacene el juego de preferencias en el vector, y otro que nos permita recuperar un juego de preferencia por su tipo. Este sería su código:

```

view plain print ?
01. public class Preferences {
02.
03.     private static Vector<Preference> listPreferences = new Vector<Preference>
04.     ();
05.
06.     public void addPreference(Preference preference) {
07.         listPreferences.addElement(preference);
08.     }
09.
10.     public static Preference getPreferenceByType(String type) {
11.         for (Preference preference:listPreferences) {
12.             if (preference.getType().equals(type)) {
13.                 return preference;
14.             }
15.         }
16.         throw new IllegalArgumentException("El tipo de preferencia no existe")
17.     }

```

Ahora es cuando utilizamos el poder de Commons Digester para mapear el XML con nuestros objetos. Se puede hacer de dos formas: la primera utilizando un API en Java y la segunda a través de un XML donde se definen las reglas de parseo. Para este ejemplo vamos a optar por hacerlo utilizando el API de Java. Entonces nos creamos la siguiente clase:

view plain print ?

```
01. package com.autentia.preferences;
02.
03. import java.io.IOException;
04.
05. import org.apache.commons.digester.Digester;
06. import org.xml.sax.SAXException;
07.
08. public class PreferencesDigester {
09.
10.     public Preferences digesterPreferences(String fileNamePreferences) throws
11.
12.         Digester digester = new Digester();
13.         digester.setValidating(false);
14.
15.         digester.addObjectCreate("preferences/preference/datasource", Datasour
16.         digester.addBeanPropertySetter("preferences/preference/datasource/url"
17.         digester.addBeanPropertySetter("preferences/preference/datasource/driv
18.         digester.addBeanPropertySetter("preferences/preference/datasource/user
19.         digester.addBeanPropertySetter("preferences/preference/datasource/pass
20.
21.         digester.addObjectCreate("preferences/preference", Preference.class);
22.         digester.addSetProperties("preferences/preference");
23.         digester.addBeanPropertySetter("preferences/preference/idioma");
24.         digester.addSetNext("preferences/preference/datasource", "setDatasourc
25.
26.         digester.addObjectCreate("preferences", Preferences.class);
27.         digester.addSetNext("preferences/preference", "addPreference");
28.
29.         Preferences preferences = (Preferences)digester.parse(ClassLoader.getS
30.
31.         return preferences;
32.
33.     }
34.
35. }
```

Como véis básicamente utilizamos cuatro funciones del API:

- addObjectCreate(patcón, clase): nos permite crear una instancia de nuestra clase a partir del patrón que le indiquemos y que tiene que coincidir con la posición en el XML de la etiqueta que va a representar la clase.
- addBeanPropertySetter(patcón): nos permite establecer el valor de la etiqueta que indica el patrón, al atributo de la clase. En caso de que el nombre de la etiqueta no coincidiera exactamente con el nombre del atributo, podríamos pasarle un segundo parámetro con el nombre del atributo donde va a establecer el valor de la etiqueta.
- addSetProperties(patcón): nos permite mapear los atributos de la etiqueta, por ejemplo, **type="default"**, a los atributos de la clase automáticamente, por lo tanto no es necesario hacerlo explícitamente uno por uno.
- addSetNext(patcón, método): nos permite indicar que método se va encargar de establecer el valor en el atributo de la clase

Una vez especificado el método de mapeo, nos basta con invocar a la función *parse* a la que le pasamos el fichero XML con las preferencias, para que aplique el mapeo y nos devuelva los objetos rellenos con los valores de las preferencias.

Para probar esto, no vamos a hacer una clase Main, lo que vamos a hacer es un test (realmente yo he ido aplicando TDD por lo que lo primero que he hecho ha sido el test). El código del test quedaría de esta forma:

```

view plain print ?
01. package com.autentia.preferences;
02.
03. import static org.junit.Assert.assertEquals;
04. import static org.junit.Assert.fail;
05.
06. import org.junit.Test;
07.
08. public class PreferencesDigesterTest {
09.
10.     @Test
11.     public void testDigesterPreferences() {
12.         try {
13.             new PreferencesDigester().digesterPreferences("preferences.xml");
14.             Preference preference = Preferences.getPreferenceByType("default")
15.             assertEquals("url-
16. default", preference.getDataSource().getUrl());
17.             assertEquals("idioma-default", preference.getIdioma());
18.             assertEquals("default", preference.getType());
19.         } catch (Exception e) {
20.             fail(e.getMessage());
21.         }
22.     }
23. }

```

Decimos que es un sistema de preferencias configurable y extensible porque ahora sin tocar una sola línea de código Java, yo puedo añadir tantos juegos de preferencias como necesite, siempre respetando la estructura del XML.

De esta forma, si ahora ya añado el siguiente juego de preferencias:

```

view plain print ?
01. <preferences>
02.     <preference type="default">
03.         <datasource>
04.             <url>url-default</url>
05.             <driver>driver-default</driver>
06.             <user>user-default</user>
07.             <password>password-default</password>
08.         </datasource>
09.         <idioma>idioma-default</idioma>
10.     </preference>
11.     <preference type="preproduccion">
12.         <datasource>
13.             <url>url-preproduccion</url>
14.             <driver>driver-preproduccion</driver>
15.             <user>user-preproduccion</user>
16.             <password>password-preproduccion</password>
17.         </datasource>
18.         <idioma>idioma-preproduccion</idioma>
19.     </preference>
20. </preferences>

```

Puedo modificar la clase de test para comprobar que efectivamente está recogiendo los dos juegos de preferencias.

view plain print ?

```
01. package com.autentia.preferences;
02.
03. import static org.junit.Assert.assertEquals;
04. import static org.junit.Assert.fail;
05.
06. import org.junit.Test;
07.
08. public class PreferencesDigesterTest {
09.
10.     @Test
11.     public void testDigesterPreferences() {
12.         try {
13.             new PreferencesDigester().digesterPreferences("preferences.xml");
14.             Preference preferenceDefault = Preferences.getPreferenceByType("de
15. default", preferenceDefault.getDatasource().getUrl());
16.             assertEquals("url-
17. default", preferenceDefault.getIdioma());
18.             assertEquals("default", preferenceDefault.getType());
19.
20.             Preference preferencePreproduccion = Preferences.getPreferenceByTy
21.             assertEquals("url-
22. preproduccion", preferencePreproduccion.getDatasource().getUrl());
23.             assertEquals("idioma-
24. preproduccion", preferencePreproduccion.getIdioma());
25.             assertEquals("preproduccion", preferencePreproduccion.getType());
26.
27.         } catch (Exception e) {
28.             fail(e.getMessage());
29.         }
30.     }
31. }
```

4. Conclusiones

No me extraña que sea una librería tan utilizada por otras, la verdad es que hace bastante fácil el mapeo de XML a clases de Java. Una pena que no haya forma (o yo no la he encontrado, si es así me ponéis un comentario) de que se puedan mapear automáticamente las etiquetas que están dentro de otras, como en el caso del datasource, de igual forma que sí lo hace con las propiedades de las etiquetas.

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)

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

COMENTARIOS



Esta obra está licenciada bajo licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5