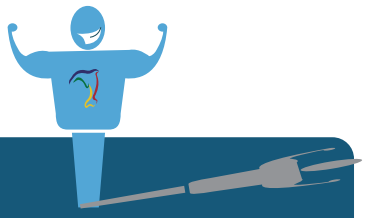
autentia

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...

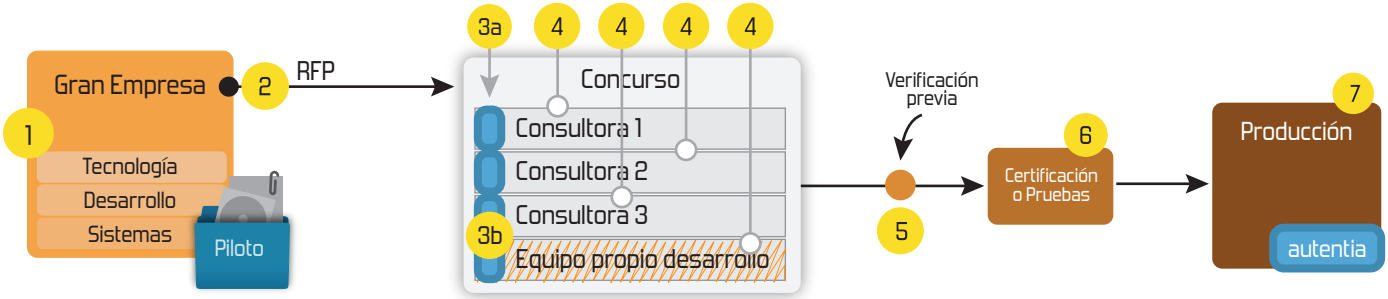
1. Desarrollo de componentes y proyectos a medida



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 conociemiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Tutorial desarrollado por

Borja Lázaro de Rafael

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero en Informática

Puedes encontrarme en Autentia

Somos expertos en Java/J2EE

Descargar este documento en formato PDF: enlazarBugzillaConMavenChangesPlugin.pdf

Fecha de creación del tutorial: 2009-09-11

Enlazar el Bugzilla con el plugin de cambios maven-changes-plugin

Índice de contenido

Introducción.

En este tutorial "Como generar con Maven un histórico de cambios del proyecto" podéis encontrar una expliación más completa de qué es el plugin de cambios de Maven (maven-changes-plugin) y como podemos configurar nuestro proyecto para utilizarlo. Por lo que resumidamente os cuento que básicamente sirve para llevar un histórico de los cambios de nuestros proyectos; saber en que versión se arregló algo, nueva funcionalidad, etc.

Actualmente el plugin "maven-changes-plugin" genera el histórico de cambios tomando como fuente un fichero XML (changes.xml), Jira, o Trac; pero de momento todavía no han sacado una versión del plugin que sea capaz de sacar el informe de Bugzilla, herramienta que utilizamos muchos de nosotros para tener un control de los cambios a implementar en nuestros proyectos.

Esto supone que para generar nuestro informe con el histórico de cambios debemos ir manualmente viendo los cambios que se han ido implementando en cada versión para crear nuestro fichero "changes.xml". Como suponéis esto es algo que podemos solucionar haciendo que el fichero changes.xml, se genere automáticamente tomando la información que hay en nuestro Bugzilla; pero aún es mejor si hacemos que esto se haga de forma transparente cada vez que generamos la documentación de nuestro proyecto con Maven. Por eso en este tutorial os vamos a contar como crear un plugin de Maven que genere automáticamente el fichero changes.xml, tomando la información del Bugzilla, para ser utilizado por el plugin maven.changes-plugin en la generación de su histórico de cambios.

Entorno

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portatil Samsung R70 (Intel(R) Core(TM)2 Duo 2,5Ghz, 2046 MB RAM, 232 Gb HD)
- Sistema Operativo: Windows Vista Home Premium
- Máquina Virtual Java: JDK 1.5.0_14 de Sun Microsystems (http://java.sun.com/javase/downloads/index_jdk5.jsp
- IDE Eclipse 3.3 (Europa) (http://www.eclipse.org/downloads/)

Creación del plugin complementario.

Como hemos explicado en la introducción, la solución que hemos adoptado es crear un plugin de Maven para que se conecte a nuestro Bugzilla obteniendo la lista de bugs y genere el fichero changes.xml antes de que lo necesite el plugin de cambios de Maven para generar el histórico de cambios.

Para ver como empezar a crear nuestro plugin podéis ver este tutorial "Desarrollo de Plugins para Maven" que explica como hacer un plugin de Maven sencillo.

En primer lugar identificamos los parámeteros que necesita nuestro plugin para conectarse al Bugzilla y generar el fichero changes.xml:

- productName: Nombre del producto en el Bugzilla del que obtenemos la lista de bugs.
- componentName: Nombre del componente en el Bugzilla del que obtendremos la lista de bugs.
- bugzillaUser: Usuario para el login en el Bugzilla.
- bugzillaPassword: Contraseña del usuario para el login en el Bugzilla.
- resolution: Estado de resolución de los bugs a tener en cuenta. (Por defecto sólo se tienen en cuenta los que tienen estado a fixed)
- loginPage: Página de login en el Bugzilla.
- xmlPath: Ruta donde se generará el fichero changes.xml que debe ser la misma que la que definamos en el maven-changes-plugin.
- loginRequired: Parámetro para indicar si es necesario hacer login en el Bugzilla para recuperar la lista de bugs. (por defecto true)
- parentOnly: Al tener los proyectos de Maven herencia, con este parámetros le indicamos que sólo genere la información de cambios en el proyecto padre.
- fitDevelopers: La información del Bugzilla nos indica quién tiene asignado el bug como una dirección de correo. Con este parámetro indicamos que la información correspondiente al desarrollador que va a ir en el "changes.xml" se ajuste al usuario de la cuenta. (p.e. si tenemos "usuario@dominio.com" se quedaría con "usuario").
- currentVersionOnly: Debido a que la lista de bugs de todas las versiones del producto puede ser muy larga, con este parámetro indicamos que sólo tenga en cuenta la versión actual (definida en el POM). Si ya tenemos un fichero changes.xml, con este parámetro lo que hacemos es añadir o sustituir los cambios de esta versión. Si está a "true" se genera la fecha de la versión con la fecha actual, para el resto de casos la fecha de la versión se deja en blanco.
- versionSuffix: El nombre de la versión es común que tenga un sufijo mientras se está desarrollando, por lo que al consultar en el Bugzilla se debe eliminar. (por defecto "-SNAPSHOT").
- componentNamePrefix2Remove: El nombre de los módulos puede que tengan un prefijo común que no están en el nombre del componente en el Bugzilla. Con este parámetro indicamos cual es dicho prefijo que será eliminado para hacer la consulta en el Bugzilla. (por defecto "").

Todos estos parámetros deben formar parte de nuestro plugin, por lo que en nuestro código debemos añadir:

view plain print ?

```
01. /**
02.  * Bugzilla product name.
03.  *
04.  * @parameter expression="${changesMavenPlugin.productName}"
05.  * @required
06.  */
07. private String productName;
08.
09. /**
10.  * Bugzilla product compnent name.
11.  *
12.  * @parameter expression="${changesMavenPlugin.componentName}" default-value="${project.artifactId}"
13.  */
14. private String componentName;
15.
16. /**
17.  * Bugzilla user.
18.  *
19.  * @parameter expression="${changesMavenPlugin.bugzillaUser}"
20.  * @required
21.  */
22. private String bugzillaUser;
23.
24. /**
25.  * Bugzilla password for the user.
26.  *
27.  * @parameter expression="${changesMavenPlugin.bugzillaPassword}"
28.  * @required
29.  */
30. private String bugzillaPassword;
31.
32. /**
33.  * Sets the resolution(s) that you want to fetch from Bugzilla. Valid resolutions are: <code>FIXED</code>,
34.  * <code>INVALID</code>, <code>WONTFIX</code>, <code>DUPLICATE</code>, <code>WORKSFORME</code>,
35.  * <code>MOVED</code> and <code>---</code>. Multiple values can be separated by commas.
36.  *
37.  * @parameter expression="${changesMavenPlugin.resolution}" default-value="FIXED"
38.  */
39. private String resolution;
40.
41. /**
42.  * Bugzilla login page.
43.  *
44.  * @parameter expression="${changesMavenPlugin.loginPage}" default-value="index.cgi"
45.  */
46. private String loginPage;
47.
48. /**
49.  * The path of the <code>changes.xml</code> file that will be generated to maven-changes-plugin. Must be the same path
50.  * configured in maven-changes-plugin.
51.  *
52.  * @parameter expression="${changesMavenPlugin.xmlPath}" default-value="src/changes/changes.xml"
53.  */
54. private File xmlPath;
55.
56. /**
57.  * Bugzilla login required.
58.  *
59.  * @parameter expression="${changesMavenPlugin.loginRequired}" default-value="true"
60.  */
61. private boolean loginRequired;
62.
63. /**
64.  * Changes report only in parent project.
65.  *
66.  * @parameter expression="${changesMavenPlugin.parentOnly}" default-value="true"
67.  */
68. private boolean parentOnly;
69.
70. /**
71.  * Fits Bugzilla developers email with Maven developers removing from @ to the end.
72.  * @parameter expression="${changesMavenPlugin.fitDevelopers}" default-value="true"
73.  */
74. private boolean fitDevelopers;
75.
76. /**
77.  * Changes report only for current version
78.  *
79.  * @parameter expression="${changesMavenPlugin.currentVersionOnly}" default-value="true"
80.  */
81. private boolean currentVersionOnly;
82.
83. /**
84.  * Version name suffix that is removed from the Bugzilla Http request.
85.  *
86.  * @parameter expression="${changesMavenPlugin.versionSuffix}" default-value="-SNAPSHOT"
87.  */
88. private String versionSuffix;
89.
90. /**
91.  * Component name prefix that is removed from the Bugzilla Http request.
92.  *
93.  * @parameter expression="${changesMavenPlugin.componentNamePrefix2Remove}" default-value=""
94.  */
95. private String componentNamePrefix2Remove;
96.
```

El proceso principal de nuestro plugin se realiza en el método "execute()" que necesariamente debe implementar al heredar de "org.apache.maven.plugin.AbstractMojo". Los pasos que sigue son:

- Preprocesamiento de parámetros: Para controlar si seguir con la ejecución y preparar los datos necesarios para el resto del proceso.
- Login en el Bugzilla.
- Recuperar la lista de bugs del Bugzilla
- Recuperar el documento XML del Bugzilla con la información de los bugs de la lista.
- Formar el "changes.xml" transformando el XML del bugzilla.

El código del método "execute()" es:

Directorio de ejemplos de jMonkey Engine

2009-07-19
JSR-179 Location API para J2ME: Posicionamiento geográfico en nuestras aplicaciones.

2009-07-16
Gestión de Usuarios en TNT Concept versión 0.16.1

2009-07-16
Continuación del Tutorial: JMonkeyEngine, Creación de nuestro primer juego.

2009-07-16
Como implementar el Scene Monitor para analizar las escenas en jMonkeyEngine

2009-02-26
Transformaciones de escena en JMonkeyEngine

2009-07-15
Detalles del juego de la moto en jMonekyEngine.

2009-07-14
JMonkeyEngine, Creación de nuestro primer juego.

2009-07-13
Ajax tests con Selenium: prototype.js e ICEfaces.

2009-07-08
AOP con AspectJ y Maven

2009-07-07
Instalación y configuración de Eclipse Galileo

2009-07-07
Iniciarse en el manejo de JME, Creación de un Cloth.

2009-07-06
Primeros pasos con Blender: Pintando nuestra mascota en 3D

2009-07-06
DBUnit-Exportar e Importar BBDD

2009-07-05
JMeter, Pruebas de stress sobre aplicaciones web: Grabando y reproduciendo navegaciones

2009-07-02
Axis2: Invocación de Servicios Web usando distintos MEP

2009-07-02
Instalación OpenOffice

2009-07-02
Juegos 3D en Java: Blender y JMonkeyEngine

2009-06-20
StAX (Xml Pull Parser): Streaming API para XML

2009-06-15
Configuración de la desconexión de usuarios con ICEFaces

2009-06-10
LWUIT: Una librería gráfica tipo AWT o Swing para J2ME

2009-06-10
Mapas mentales con XMind

2009-02-26
Redimensionar Imagenes en Windows Vista

2009-06-08
UploadFile con Icefaces + Hibernate + Anotaciones

2009-06-05
Habilitar exportación en Liferay

2009-06-01
Registrar Liferay en Eclipse

2009-05-29
Liferay Social Office

2009-05-28
Broadcast con Ustream

2009-05-25
Tabla datos accesible con ordenación y

view plain print ?

```
/*
 * (non-Javadoc)
 *
 * @see org.apache.maven.plugin.AbstractMojo#execute()
 */
public void execute() throws MojoExecutionException, MojoFailureException {

    this.getLog().debug("======= ENTERING");
    this.getLog().info("component:" + this.componentName);

    // si el informe el solo para el padre y es un hijo salimos
    // si es un hijo y no está definido el component salimos
    if ((this.parentOnly && (this.project.getParent() != null))
            || (!this.parentOnly && ((this.componentName == null) || this.componentName.equals("")))) {
        return;
    }

    // recuperamos el nombre de la version
    this.versionName = this.project.getVersion();
    final int index = this.versionName.indexOf(this.versionSuffix);
    if (index != -1) {
        // quitamos el sufijo
        this.versionName = this.versionName.substring(0, index);
    }

    // quitamos el prefijo del nombre del componente
    if ((this.componentNamePrefix2Remove != null) && !this.componentNamePrefix2Remove.equals("")) {
        if (this.componentName.startsWith(this.componentNamePrefix2Remove)) {
            this.componentName = this.componentName.substring(this.componentNamePrefix2Remove.length());
        }
    }
    // inicializamos el gestor de peticiones
    this.httpRequest = new HttpRequest(this.getLog());
    // inicializamos la url del Bugzilla
    this.bugzillaURL = this.project.getIssueManagement().getUrl();
    // preparamos el cliente HTTP para las peticiones
    final HttpClient client = new HttpClient();
    final HttpClientParams clientParams = client.getParams();
    clientParams.setBooleanParameter(HttpClientParams.ALLOW_CIRCULAR_REDIRECTS,true);
    final HttpState state = new HttpState();
    final HostConfiguration hc = new HostConfiguration();
    client.setHostConfiguration(hc);
    client.setState(state);
    this.determineProxy(client);

    // hacemos el login si es necesario
    boolean loginSuccess = true;
    if (this.loginRequired) {
        loginSuccess = this.login(client);
    }

    if (loginSuccess) {
        // recuperamos la lista de bugs
        String bugsIds;
        bugsIds = this.getBugList(client);

        // recuperamos el xml de bugs
        Document bugsDocument;
        bugsDocument = this.getBugsDocument(client, bugsIds);

        // formamos el changes.xml
        this.builChangesXML(bugsDocument);

    }
    this.getLog().debug("======= EXITING");
}

```

Como se puede ver en el código, después de un procesamiento previo de los parámetros se empieza con las peticiones al Bugzilla; hacemos el login y realizamos la petición que nos devuelve la lista de Bugs. Esta petición es del estilo "https://host/cgi-bin/bugzilla3/buglist.cgi?product=productName&resolution=FIXED&order=target_milestone" que devuelve una página Web en HTML; que deberemos procesar para poder sacar los identificadores de los bugs que queremos recuperar. Si vemos el código HTML de la respuesta podemos ver un formulario como este:

view plain print ?

```
<form action="show_bug.cgi" method="post">
     <input type="hidden" value="xml" name="ctype"/>
     <input type="hidden" value="1" name="id"/>
     <input type="hidden" value="2" name="id"/>
     <input type="hidden" value="3" name="id"/>
     ......
     <input type="hidden" value="N" name="id"/>
     <input type="hidden" value="attachmentdata" name="excludefield"/>
     <input type="submit" id="xml" value="XML"/>
 </form>
```

Si nos fijamos en el código HTML completo, tenemos únicamente dos formularios con "action=show_bug.cgi", pero los id's que se envían son siempre los mismos; por lo que procesando la respuesta HTML recuperaremos todos los id's que tenemos que enviar para recuperar la petición. Como la respuesta es HTML, y para poder procesarla mejor, la convertimos a XHTML utilizando JTidy; para lo que en el POM.xml debemos añadir la siguiente dependencia:

view plain print ?

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>jtidy-r8</artifactId>
    <version>21122004</version>
</dependency>

```

Para convertir un String que representa a un HTML en un documento XML lo hacemos con el siguiente código:

view plain print ?

```
final Tidy tidy = new Tidy();
tidy.setXHTML(true);
tidy.setMakeClean(true);
tidy.setBreakBeforeBR(true);
tidy.setTidyMark(false);
tidy.setQuoteAmpersand(false);
tidy.setQuoteMarks(false);
tidy.setQuoteNbsp(false);
tidy.setRawOut(true);
tidy.setFixComments(true);
tidy.setSmartIndent(true);
tidy.setWraplen(4000);
tidy.setDocType("omit");
tidy.setShowWarnings(false);
tidy.setQuiet(true);
tidy.setIndentAttributes(false);
tidy.setIndentContent(false);
tidy.setSpaces(0);
tidy.setTabsize(0);
final ByteArrayOutputStream baos = new ByteArrayOutputStream();
final ByteArrayInputStream bais = new ByteArrayInputStream(response.getBytes());
final Document doc = tidy.parseDOM(bais, baos);

```

Habiendo procesado el formulario, enviamos la petición al Bugzilla para que nos devuelva el XML con la lista de bugs. Es muy importante que no nos olvimemos que la petición la debemos realizar por POST; y que no nos podemos olvidar de los parámetos "ctype=xml" (indica que queremos la respuesta en formato XML) y "excludefield=attachmentdata" (para que no nos incluya los fichero adjuntos a los bugs).

Debido a los posible tipos de codificación que puede haber entre el servidor del Bugzilla y nuestra JVM os aconsejo que esta respuesta en particular la tratemos siempre que podamos en bytes; por lo que en la clase que realizamos las peticiones HTTP tendremos el siguiente método:

view plain print ?

```
01. /**
02.  * Send a GET method request to the given link using the configured HttpClient, possibly following redirects, and returns
03.  * the response as String.
04.  *
05.  * @param cl the HttpClient
06.  * @param link the URL
07.  * @throws HttpStatusException
08.  * @throws IOException
09.  */
10. public byte[] sendPostRequest(final HttpClient cl, final String link, final String parameters)
11.         throws HttpStatusException, IOException {
12.     try {
13.         final PostMethod pm = new PostMethod(link);
14.
15.         if (parameters != null) {
16.             final String[] params = parameters.split("&");
17.             for (final String param : params) {
18.                 final String[] pair = param.split("=");
19.                 if (pair.length == 2) {
20.                     pm.addParameter(pair[0], pair[1]);
21.                 }
22.             }
23.         }
24.
25.         this.getLog().info("Downloading from Bugzilla at: ' + link);
26.
27.         cl.executeMethod(pm);
28.
29.         final StatusLine sl = pm.getStatusLine();
30.
31.         if (sl == null) {
32.             this.getLog().error("Unknown error validating link: ' + link);
33.
34.             throw new HttpStatusException("UNKNOWN STATUS");
35.         }
36.
37.         // if we get a redirect, throws exception
38.         if (pm.getStatusCode() == HttpStatus.SC_MOVED_TEMPORARILY) {
39.             this.getLog().debug("Attempt to redirect ");
40.             throw new HttpStatusException("Attempt to redirect");
41.         }
42.
43.         if (pm.getStatusCode() == HttpStatus.SC_OK) {
44.             final InputStream is = pm.getResponseBodyAsStream();
45.             final ByteArrayOutputStream baos = new ByteArrayOutputStream();
46.             final byte[] buff = new byte[256];
47.             int readed = is.read(buff);
48.             while (readed != -1) {
49.                 baos.write(buff, 0, readed);
50.                 readed = is.read(buff);
51.             }
52.
53.             this.getLog().debug("Downloading from Bugzilla was successful");
54.             return baos.toByteArray();
55.         } else {
56.             this.getLog().warn("Downloading from Bugzilla failed. Received: [ + pm.getStatusCode() + "]");
57.             throw new HttpStatusException("WRONG STATUS");
58.         }
59.     } catch (final HttpException e) {
60.         if (this.getLog().isDebugEnabled()) {
61.             this.getLog().error("Error downloading issues from Bugzilla:', e);
62.         } else {
63.             this.getLog().error("Error downloading issues from Bugzilla url:  + e.getLocalizedMessage());
64.
65.         }
66.         throw e;
67.     } catch (final IOException e) {
68.         if (this.getLog().isDebugEnabled()) {
69.             this.getLog().error("Error downloading issues from Bugzilla:', e);
70.         } else {
71.             this.getLog().error("Error downloading issues from Bugzilla. Cause is  + e.getLocalizedMessage());
72.         }
73.         throw e;
74.     }
75. }
76.
```

Y en nuestra clase principal, formaremos el documento XML con el siguiente código:

view plain print ?

```
01. .....
02. final byte[] response = this.httpRequest.sendPostRequest(client, link, bugsIds);
03.
04. final DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
05. final DocumentBuilder db = dbf.newDocumentBuilder();
06. db.setEntityResolver(new EntityResolver() {
07.
08.     public InputSource resolveEntity(final String publicId, final String systemId) throws SAXException,
09.             IOException {
10.         return new InputSource(this.getClass().getClassLoader().getResourceAsStream(
11.                 "bugzilla3/bugzilla.dtd'));
12.     }
13. });
14. final ByteArrayInputStream bais = new ByteArrayInputStream(response);
15.
16. final Document docBugzilla = db.parse(bais);
17. .....
18.
```

Donde se puede ver que formamos el documento XML directamente con los bytes que nos devuelve el servidor del Bugzilla. También cambiamos el "EntityResolver" para que independientemente del DTD que vaya definido en el XML, que puede no estar accesible provocando un fallo, lo recupere de nuestros recursos.

Habiendo obtenido el documento XML con los bugs, lo sometemos a un procesamiento previo que facilite su posterior transformación. Al elemento "target_milestone" le añadimos 2 atributos que faciliten la ordenación en su transformación. Por ejemplo, para un <target_milestone>1.13</target_milestone> se convertiría en <target_milestone version1="1" version2="13">1.13</target_milestone>. También aprovechamos para eliminar todos aquellos nodos que no nos van a hacer falta en la transformación XSL.

Ahora ya tenemos el documento XML del bugzilla con la información de los bugs que necesitamos, por lo que vamos a transformarlo al formato del "changes.xml". Para hacer esto utilizamos el siguiente XSL:

view plain print ?

```
01. <xsl:stylesheet version="2.0"
02.     xmlns:xsl="http://www.w3.org/1999/XSL/Transform'
03.     xmlns:fo="http://www.w3.org/1999/XSL/Format'
04.     xmlns:xs="http://www.w3.org/2001/XMLSchema'
05.     xmlns:fn="http://www.w3.org/2005/xpath-functions'>
06.     <xsl:output method="xml" indent="yes" encoding="UTF-8" />
07.     <xsl:key name="version" match="version" use="version" />
08.     <xsl:template match="/bugzilla">
09.         <document>
10.             <properties>
11.                 <title>
12.                     <xsl:value-of select="title" />
13.                 </title>
14.             </properties>
15.             <body>
16.                 <!-- recuperamos la fecha de la versión --
17.                 <xsl:variable name="date">
18.                         <xsl:value-of select="date" />
19.                 </xsl:variable>
20.                 <xsl:for-each
21.                     select="bug[not(target_milestone=preceding-sibling::bug/target_milestone)]>
22.                     <xsl:sort select="target_milestone/@version1' data-type="number" />
23.                     <xsl:sort select="target_milestone/@version2' data-type="number" />
24.                     <xsl:variable name="version">
25.                         <xsl:value-of select="target_milestone" />
26.                     </xsl:variable>
27.                     <release>
28.                         <xsl:attribute name="version"><xsl:value-of
29.                                 select="$version" />
30.                         </xsl:attribute>
31.                         <xsl:attribute name="date"><xsl:value-of
32.                                 select="$date" />
33.                         </xsl:attribute>
34.                         <xsl:for-each select="/bugzilla/bug[target_milestone = $version]>
35.                             <xsl:sort select="bug_id" data-type="text"/>
36.                             <action>
37.                                 <xsl:attribute name="type">
38.                                     <xsl:choose>
39.                                         <xsl:when test="bug_severity = 'enhancement''>add</xsl:when>
40.                                         <xsl:otherwise>fix</xsl:otherwise>
41.                                     </xsl:choose>
42.                                 </xsl:attribute>
43.                                 <xsl:attribute name="issue">
44.                                     <xsl:value-of select="bug_id" />
45.                                 </xsl:attribute>
46.                                 <xsl:attribute name="dev">
47.                                     <xsl:value-of select="assigned_to" />
48.                                 </xsl:attribute>
49.                                 <xsl:value-of select="short_desc" />
50.                             </action>
51.                         </xsl:for-each>
52.                     </release>
53.                 </xsl:for-each>
54.             </body>
55.         </document>
56.     </xsl:template>
57. </xsl:stylesheet>
58.
```

En el XSL se puede observar la particualridad de formar los elementos <release>. Como los bugs del documento del Bugzilla pueden venir desordenados, al recorrerlos en el bucle de la línea 18, se indica que sólo se coja un bug de cada versión; de esta forma obtenemos la información para formar por completo el elemento <release>. Y recorriendo este bucle ya procesamos todos los bugs de la versión en bucle de la línea 35; donde formaremos la información específica de cada bug.

Esta información se corresponde con:

- El título lo pondremos como el título del proyecto, que se añadirá como un elemento más al XML del Bugzilla.
- El nombre de la versión con el valor del elemento <target_milestone>.
- El tipo de cada bug será "add" si en el Bugzilla está como mejora, y "fix" en cualquier otro caso.
- El identificador del bug con el valor del elemento <bug_id>.
- El desarrollador con el valor del elemento <assigned_to>.
- La descripción del bug con el valor del elemento <short_desc>.

Por último sólo nos queda hacer que nuestro plugin pueda ser utilizado desde otros proyectos instalandolo en nuestro repositorio local ejecutando:

```
mvn clean install
```

Utilizando nuestro plugin.

Como hemos dicho nuestro plugin es un complemento para el plugin de informes de cambios maven-changes-plugin. Éste plugin de cambios se ejecuta en la fase de maven "site". Por lo que el plugin que acabamos de crear lo debemos configurar para que se ejecute justo antes, es decir en la fase "pre-site". De esta forma en el "pom.xml" de nuestros proyectos será del estilo:

view plain print ?

```
01. <project ...>
02.   .....
03.   <issueManagement>
04.     <system>Bugzilla</system>
05.     <url>https://host/cgi-bin/bugzilla3/</url>
06.   </issueManagement>
07.   .....
08.   <build>
09.         <plugins>
10.       .....
11.             <plugin>
12.               <!-- nuestro plugin de generación del changes.xml --
13.                 <groupId>com.autentia.mvn.plugin</groupId>
14.                 <artifactId>bugzillaChanges</artifactId>
15.                 <configuration>
16.                     <productName> *** nombre del producto en el bugzilla ***</productName>
17.                     <bugzillaUser>${bugzillaUser}</bugzillaUser>
18.                     <bugzillaPassword>${bugzillaPassword}</bugzillaPassword>
19.                     <currentVersionOnly>false</currentVersionOnly>
20.                 </configuration>
21.               <executions>
22.                     <execution>
23.                         <phase>pre-site</phase>
24.                         <goals>
25.                             <goal>changes-generate</goal>
26.                         </goals>
27.                     </execution>
28.                 </executions>
29.             </plugin>
30.         </plugins>
31.   </build>
32.   .....
33.   <reporting>
34.     <plugins>
35.         .....
36.         <plugin>
37.             <groupId>org.apache.maven.plugins</groupId>
38.             <artifactId>maven-changes-plugin</artifactId>
39.             <version>2.0</version>
40.             <configuration>
41.                 <issueLinkTemplate>%URL%/show_bug.cgi?id=%ISSUE%</issueLinkTemplate>
42.             </configuration>
43.             <reportSets>
44.                 <reportSet>
45.                     <reports>
46.                         <report>changes-report</report>
47.                     </reports>
48.                 </reportSet>
49.             </reportSets>
50.         </plugin>
51.         .....
52.     </plugins>
53.   </reporting>
54.   .....
55.   <!-- Repositorio para descargar el plugin --
56.   <pluginRepositories>
57.       <pluginRepository>
58.           <id>AutentiaBugzillaMaven-release</id>
59.           <name>Local Maven repository of release</name>
60.           <url>http://bugzillachanges.sourceforge.net/maven-repositor</url>
61.           <snapshots>
62.               <enabled>false</enabled>
63.           </snapshots>
64.           <releases>
65.               <enabled>true</enabled>
66.           </releases>
67.       </pluginRepository>
68.   </pluginRepositories>
69.   .....
70. </project>
71.
```

De los parámetros de configuración de nuestro plugin, se pueden considerar un poco distintos los parámetros "bugzillaUser" y "bugzillaPassword", ya que es algo particular de cada usuario. Al ser el "pom.xml" un fichero que se suele compartir, no es aconsejable que aparezcan estos datos; por lo que nos valemos de las ventajas de Maven para recuperarlos del perfil activo en el fichero "settings.xml"; de esta forma en nuestro fichero "setting.xml" deberá aparecer:

view plain print ?

```
01. <settings>
02.     <profiles>
03.       .....
04.       <profile>
05.          <id>identificador del perfil activ</id>
06.          .....
07.          <properties>
08.               <bugzillaUser>user</bugzillaUser>
09.               <bugzillaPassword>password</bugzillaPassword>
10.           </properties>
11.       </profile>
12.       .....
13.     </profiles>
14.     .....
15. </settings>
16.
```

Conclusiones

Ya sabemos que gracias a Maven tenemos una gestión de nuestros proyectos más completa, y que con una sola herramienta podemos unificar varias tareas de la gestión de proyectos; pero a veces no se ajusta exáctamente a lo que necesitamos. Es aqui donde la extensibilidad de Maven por medio de plugins nos proporciona esa posibilidad que nos facilita la vida.

Animaros a hacer plugins de Maven, ya que combinando la potencia de varios plugins al final podemos conseguir lo que buscamos.

Si queréis todo el código fuente de este plugin lo podéis conseguir en sourceforge.

Documentación extra la podéis encontrar aqui.

Un saludo.
Borja Lázaro de Rafael.

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.

Nombre: E-Mail:

Comentario:

Enviar comentario **Texto Legal y condiciones de uso**

- Puedes inscribirte en nuestro servicio de notificaciones haciendo clic aquí.
- Puedes firmar en nuestro libro de visitas haciendo clic aquí.
- Puedes asociarte al grupo AdictosAlTrabajo en XING haciendo clic aquí.
- Añadir a favoritos Technorati.

Recuerda

Autentia te regala la mayoría del conocimiento aquí compartido (Ver todos los tutoriales). Somos expertos en: J2EE, Struts, JSF, C++, OOP, UML, UP, Patrones de diseño ... y muchas otras cosas.

¿Nos vas a tener en cuenta cuando necesites consultoría o formación en tu empresa?, ¿Vas a ser tan generoso con nosotros como lo tratamos de ser con vosotros?

Somos pocos, somos buenos, estamos motivados y nos gusta lo que hacemos ...

Autentia = Soporte a Desarrollo & Formación.

info@autentia.com

soluciones

Tutoriales recomendados

Nombre	Resumen	Fecha	Visitas	Valoración	Votos	Pdf
Release Bugzilla Maven Plugin	En este tutorial vamos a mostrar como automatizar un conjunto de acciones que hay que hacer siempre en los sistemas de gestión de incidencias, tales como dar de alta una nueva versión del producto, cerrar las incidencias que soluciona la nueva versión, et	2009-09-11	6	-	-	
Sobre las reglas de codificación o... ¿de dónde salen esos caracteres "raros"?.	En este tutorial vamos a tratar de dar algo de luz a los errores que tenemos habitualmente con la codificación de caracteres en aplicaciones en las que se ven implicados varios sistemas que intercambian o almacenan información.	2009-09-08	211	Muy bueno	3	
Cómo hacer deploy del site de Maven en SourceForge	Este tutorial nos enseña un poco mas sobre Maven	2009-08-28	303	Muy bueno	1	
Ordenación por cantidades en informe cruzado	Nico nos explica en ese tutorial cómo lograr ordenar por cantidades en informes cruzados usando JasperReports e iReport	2009-08-26	373	Muy bueno	4	
AOP con AspectJ y Maven	Programacion orientada a aspectos con AspectJ y Maven	2009-07-08	1005	Bueno	2	
DBUnit-Exportar e Importar BBDD	DBUnit como complemento de los test unitarios con carga a una base de datos	2009-07-06	1261	Muy bueno	6	
JMeter, Pruebas de stress sobre aplicaciones web: Grabando y reproduciendo navegaciones	En este tutorial Carlos García nos enseñará a grabar y reproducir navegaciones con JMeter, para poder realizar pruebas de carga o stress sobre aplicaciones Web	2009-07-05	2216	Bueno	10	
Configuración de la desconexión de usuarios con ICEFaces	Este tutorial muestra la manera de configurar y traducir la ventana de desconexión o pérdida de sesión del usuario en ICEFaces.	2009-06-15	2045	Muy bueno	11	
Tabla datos accesible con ordenación y paginación	En este tutorial vamos a ver como podemos hacer una tabla de datos con ordenación y paginación aplicando los criterios de accesibilidad y los conceptos de mejora progresiva y Javascript no obstrusivo.	2009-05-25	3343	Bueno	15	
Plugin Hibernate3 para Maven	En este tutorial veremos las posibilidades que nos ofrece el plugin de Hibernate3 para Maven, como por ejemplo, la generación del esquema de base de datos desde clases con anotaciones.	2009-05-02	1507	Bueno	8	

Nota: