Buscador de Contenido. Lucene.

Autor:
Sergio Raposo Vargas

Versión
1.1.

Fecha de última revisión
10/04/2007

Buscador de Contenido. Lucene.

ÍNDICE

Buscador de Contenido. Lucene.

1.- Motivación

Uno de los elementos más importantes de un portal es un buscador de contenido que permita a nuestros visitantes encontrar información rápidamente, sin necesidad de navegar por él. Si deseamos un artículo en concreto de una web con una gran cantidad de información, siendo la única forma de dar con ella navegar por indefinidas páginas, es posible que se desista de la búsqueda. Esto puede suponer que el usuario no vuelva a la página para realizar una nueva consulta.

Un buscador precisamente ofrece la herramienta necesaria para aquellos visitantes que saben muy bien que es lo que necesitan, y quieren acceder a dicha información lo más rápido posible.

OpenCms, nos ofrece una herramienta para facilitarnos el desarrollo de este tipo de utilidades, y con una programación básica podremos llegar a tener una potente solución a este tipo de problemas. Para ello, los desarrolladores de OpenCms, han apostado por Lucene. Éste es un indexador muy usado en numerosos proyectos, respaldado por la fundación Apache, y por consiguiente, nos ofrece una gran garantía.

A lo largo de este tutorial intentaré explicar todo lo referente a buscadores, y a su forma de utilizarlo con OpenCms. Pero vamos a entrar ya en materia.

2.- Teoría

Un buscador debe encontrar cualquier tipo de contenido en nuestro portal, pensad que esta búsqueda la hacemos directamente sobre los contenidos recorriendo todos y cada uno de ellos, mostrando los que coincidan con los criterios de búsqueda. De esta forma, hemos implementado un buscador, pero probablemente, el tiempo de respuesta sería tan elevado, que haría desesperarse a cualquiera.

Lucene ha generado una solución a este problema, de forma que almacena, en lo que han denominado Índices, todo el contenido pero estructurado de forma que recorrerlo y buscar la información se realiza de una forma rápida y sencilla. A esto le unimos que estos índices se almacenan en disco, y que por tanto el acceso es bastante rápido.

De normal, debemos guardar en estos índices todo lo que queramos buscar posteriormente, como por ejemplo, el contenido, el autor, las fechas, etc.

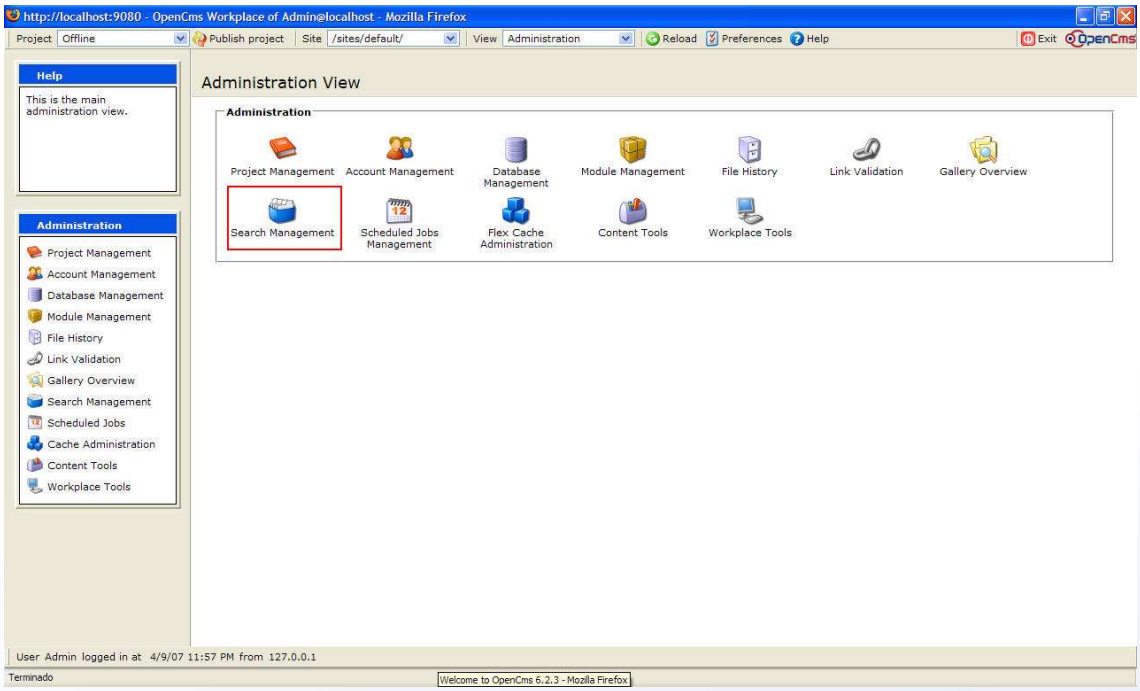
Pero nos queda una cuestión, ¿Cuándo se actualizan estos índices? Pues debemos actualizarlos cada vez que se crea, o modifica un recurso, de forma que el contenido del buscador esté siempre actualizado. El funcionamiento básico es el siguiente, existirá un listener (escuchador) pendiente de escuchar todas las acciones que se realizan sobre el contenido, de forma que actualiza la información que tiene almacenada. Lucene también permite la opción de generar desde cero dichos índices, ya que, debido a comportamientos anómalos se pueden corromper, y quedarse inutilizados.

3.- Prepara OpenCms

El primer paso que debemos dar para crear nuestro buscador, es preparar nuestros Índices.

Si, en la vista de administración nos situamos sobre la opción de Search Management, veremos un listado de los índices creados. Por defecto, dispondremos de 3:

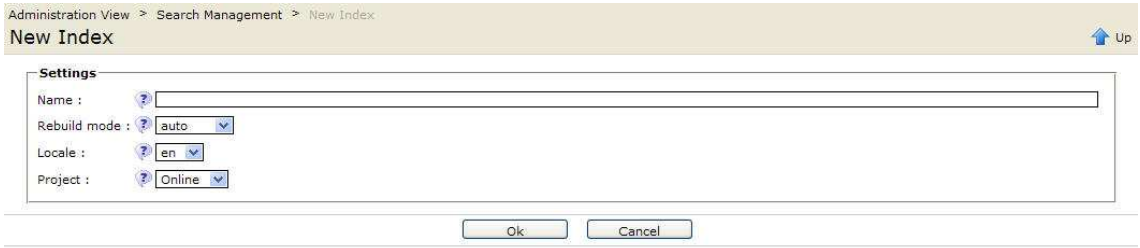
- German online help
- Offline Project
- Online Project



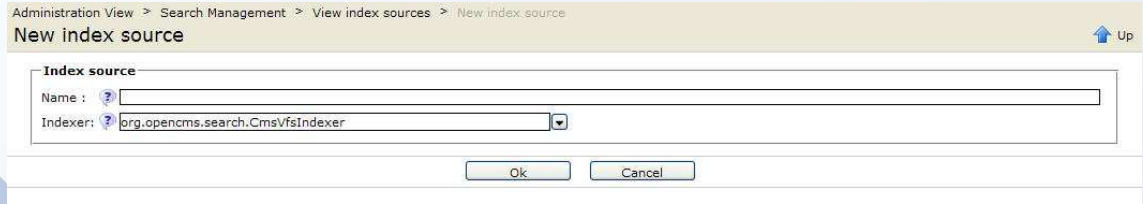
Buscador de Contenido. Lucene.

Es aconsejable crear nuestros propios índices, independientes de los ya creados. Para ello, usamos la opción New Index, y rellenamos los siguientes campos:

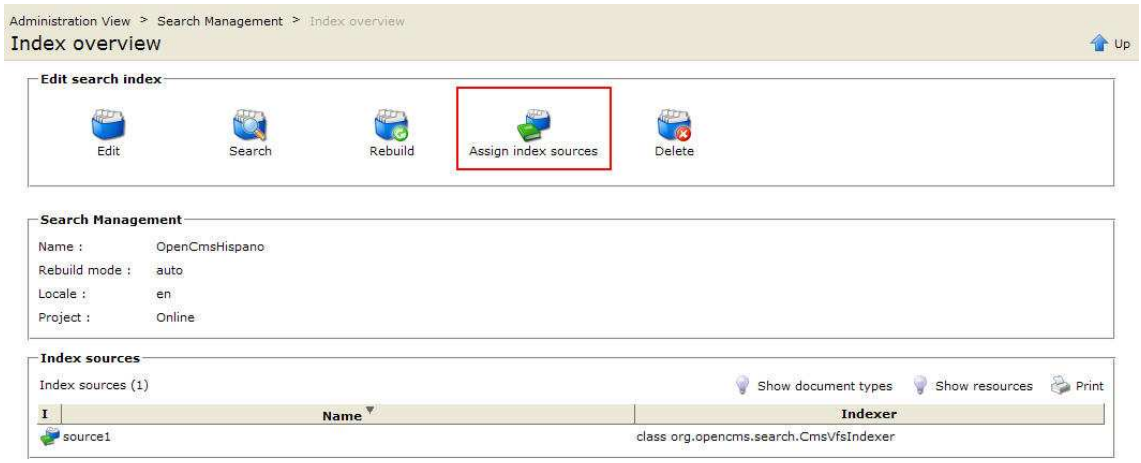
- Name: Nombre del índice. Es aconsejable usar un nombre corto pero descriptivo.
- Rebuild mode: Indicaremos auto para que se actualicen automáticamente lo índices.
- Locale: Marcamos el locale asociado a dichos indices.
- Project: Marcamos el proyecto, que para nuestro buscador, será Online.



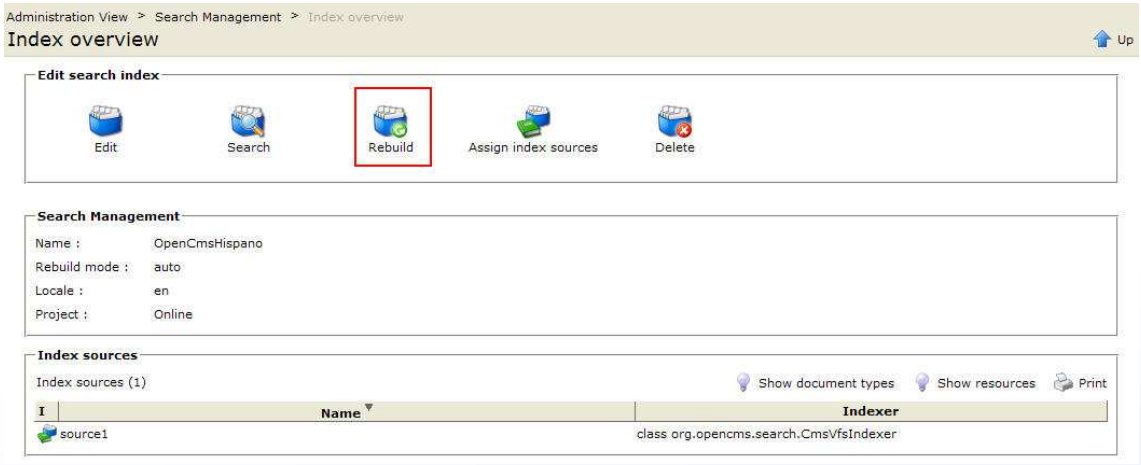
A continuación debemos crear un "source" para nuestro portal, en el que le indicaremos la ruta a partir de la cual vamos a indexar el contenido, y los tipos de contenidos que queremos indexar (esto dependerá de nuestras necesidades en cada caso). Para esto, seleccionamos View index source/New index source.



Llegado a este punto, debemos asociar el source creado, con el índice anterior, para ello pinchamos sobre el índice y seleccionamos la opción Assign index source.



Por último, debemos generar el índice por primera vez, para ello existe la opción de Rebuild con la que construiremos los índices por primera vez.



4.- Elementos del buscador

Una vez que tenemos el índice creado, debemos crear nuestra estructura de templates para implementar nuestro buscador.

Como siempre, desde OpenCmsHispano apostamos por crear una estructura de la siguiente forma:

- (System)Modulo frontend
  - Template
    - Resultado.jsp
    - Búsqueda_avanzada.jsp
    - Menu/Cabecera (para búsqueda simple).
- (Site)Microsite
  - Buscador
    - Búsqueda_avanzada.html
    - Resultado.html

A continuación escribo el código de los templates:

Búsqueda_avanzada.jsp

```
<div class="listadoNoticias">
	<div class="tituloPrincipalInterior">Busqueda Avanzada</div>
	<div class="bordeTopSep"></div>
	<%
	// Create a JSP action element
	org.opencms.jsp.CmsJspActionElement cms = new
CmsJspActionElement(pageContext, request, response);

	// Get the search manager
	CmsSearchManager searchManager = OpenCms.getSearchManager();

	%>
	<jsp:useBean id="search" scope="request"
class="org.opencms.search.CmsSearch">

		<jsp:setProperty name = "search" property="*"/>
		<%
			search.init(cms.getCmsObject());
		%>
	</jsp:useBean>
	<form method="post" action="resultado.html">
	<div class="letraDescripcion">
	<table>
		<tr>
			<th valign="top">Palabra clave:</th>
			<td><input type="text" name="query" size="50"
value="<%=(cms.property("query") != null) ? cms.property("query") :
""%>"></td>
		</tr>
		<tr>
			<th valign="top">Resultados por p&aacute;gina:</th>
```

```
                        <td>
                              <select name="matchesperpage">
                              <% for(int i=5; i<21; i++) { %>
                                    <option value="<%=i%>"><%=i%></option>
                              <% } %>
                              </select>
                        </td>
                  </tr>
                  <tr>
                        <th valign="top">N&uacute;mero de p&aacute;ginas:</th>
                        <td>
                              <select name="displaypages">
                              <% for(int j=1; j<11; j++) { %>
                                    <option value="<%=j%>"><%=j%></option>
                              <% } %>
                              </select>
                        </td>
                  </tr>
                  <tr><th valign="top">Buscar en:</th><td>
                  <input type="checkbox" name="field" value="title"
                        <%=(cms.property("field") == null ||
cms.property("field").indexOf("title") >= 0) ? "checked" : "" %>>Titulo<br>
                  <input type="checkbox" name="field" value="content"
                        <%=(cms.property("field") == null ||
cms.property("field").indexOf("content") >= 0) ? "checked" : ""
%>>Contenido<br>
                  </td></tr>
            </table>
            <input type="hidden" name="index" id="index" value="OpenCmsHispano"/>
            <input type="submit" value="Submit">
            </div>
            </form>
</div>
```

Resultados.jsp

```
<div class="listadoNoticias">
      <div class="tituloPrincipalInterior">Resultado de Busqueda</div>
      <div class="bordeTopSep"></div>
      <%
      // Create a JSP action element
      org.opencms.jsp.CmsJspActionElement cms = new
CmsJspActionElement(pageContext, request, response);

      // Get the search manager
      CmsSearchManager searchManager = OpenCms.getSearchManager();

      %>
      <jsp:useBean id="search" scope="request"
class="org.opencms.search.CmsSearch">
            <jsp:setProperty name = "search" property="matchesPerPage"
param="matchesperpage"/>
            <jsp:setProperty name = "search" property="displayPages"
param="displaypages"/>
            <jsp:setProperty name = "search" property="*"/>
            <%
                  search.init(cms.getCmsObject());
            %>
      </jsp:useBean>

      <%
            int resultno = 1;
            int pageno = 0;
            if (request.getParameter("searchPage")!=null) {
                  pageno =
```

```
Integer.parseInt(request.getParameter("searchPage"))-1;
            }
            resultno = (pageno*search.getMatchesPerPage())+1;

            String fields = search.getFields();
         if (fields==null) {
            fields = request.getParameter("fields");
         }

         List result = search.getSearchResult();
         if (result == null) {
      %>
      <%
                  if (search.getLastException() != null) {
      %>
      <h3>Error</h3>
      <%= search.getLastException().toString() %>
      <%
                  }

            } else {

                  ListIterator iterator = result.listIterator();
      %>
      <div class="letraTitulo"><%= search.getSearchResultCount() %>
Resultados para la consulta &lt;<%= search.getQuery() %>&gt; </div><br/><br/>
      <%
                  while (iterator.hasNext()) {
                        CmsSearchResult entry =
(CmsSearchResult)iterator.next();

      if(resultno%2==0)
                  {
            %>
                  <div class="noticiaSinFondo"><%
                  }else{
            %>
                  <div class="fondoNoticiaAzul"><%
                  }
      %>
                        <div class="contenidoNoticiaSinFoto">

                              <div class="letraTitulo"><%= resultno
%>. <%= entry.getTitle() %> (<%= entry.getScore() %>%)</div>
                              <div class="letraDescripcion"><%=
entry.getExcerpt() %></div>

                              <div class="botonDerSin">
                                    <a href="<%=
cms.link(cms.getRequestContext().removeSiteRoot(entry.getPath())) %>"><img
border="0" class="inline" src="<cms:link>../resources/masInfo.gif</cms:link>"
alt="Más información..."></img></a>
                              </div>
                        </div>
                  </div>
      <%
                  resultno++;
                  }
            }
      %>
      <%
            if (search.getPreviousUrl() != null) {
      %>
            <a href="<%= cms.link(search.getPreviousUrl()) %>&fields=<%=
fields %>">&lt;&lt;</a>
```

```
	<%
		}
		Map pageLinks = search.getPageLinks();
		Iterator i =  pageLinks.keySet().iterator();
		while (i.hasNext()) {
			int pageNumber = ((Integer)i.next()).intValue();
			String pageLink = cms.link((String)pageLinks.get(new
Integer(pageNumber)));
			out.print("   ");
			if (pageNumber != search.getSearchPage()) {
	%>
				<a href="<%= pageLink %>&fields=<%= fields %>"><%=
pageNumber %></a>
	<%
			} else {
	%>
				<span class="currentpage"><%= pageNumber %></span>
	<%
			}
		}
		if (search.getNextUrl() != null) {
	%>
			   
			<a href="<%= cms.link(search.getNextUrl()) %>&fields=<%=
fields %>">&gt;&gt;</a>

	<%
		}

	%>
</div>
```

Una vez creado los templates, debemos asignar a nuestros html dichas plantillas y para ello, como siempre, en la propiedad template del .html, indicamos la ruta de los templates.

NOTA: Tened en cuenta que las jsp mostradas en este tutorial están preparadas para el portal OpenCmsHispano. A nivel de presentación deberéis cambiar todo lo que os sea necesario, junto al nombre del índice, etc.

5.- Buscador personalizado

Debemos tener presente que OpenCms indexa todo el contenido por defecto. Cuando hablamos del xmlcontent, no distingue de campos, sino que indexa todo el contenido en un mismo campo (content). Es posible que dentro de nuestras necesidades tengamos que indexar más información, o que debamos indexar cada campo por separado.
Para ello, debemos programar de nuevo la clase que se encarga de realizar esta tarea. En la clase org.opencms.search.documents.CmsDocumentXmlContent, se implementa el tratamiento que se realiza al contenido XmlContent. Nuestra tarea no es más que retocar esta clase para añadirle nuestra nueva funcionalidad, para ello, la copiamos, creamos un paquete propio de trabajo (para evitar siempre que sea posible tocar el core del producto) y creamos nuestra nueva clase.
A continuación pongo el código de la clase retocada que recorre todos los campos del contenido, y lo indexa, creando los campos con el prefijo xml-, y a continuación el nombre del campo.

org.opencmshispano.search.documents.CmsDocumentXmlContentHispano

```
package org.opencmshispano.search.documents;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Locale;

import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.opencms.cache.Messages;
import org.opencms.file.CmsFile;
import org.opencms.file.CmsObject;
import org.opencms.file.CmsResource;
import org.opencms.file.types.CmsResourceTypeXmlContent;
import org.opencms.file.types.I_CmsResourceType;
import org.opencms.i18n.CmsLocaleManager;
import org.opencms.main.CmsException;
import org.opencms.main.OpenCms;
import org.opencms.search.A_CmsIndexResource;
import org.opencms.search.CmsIndexException;
import org.opencms.search.documents.A_CmsVfsDocument;
import org.opencms.search.extractors.CmsExtractionResult;
import org.opencms.search.extractors.I_CmsExtractionResult;
import org.opencms.util.CmsStringUtil;
import org.opencms.xml.A_CmsXmlDocument;
import org.opencms.xml.content.CmsXmlContentFactory;
import org.opencms.xml.types.I_CmsXmlContentValue;

/**
 * Lucene document factory class to extract index data from a cms resource
 * of type <code>CmsResourceTypeXmlContent</code>.<p>
 *
```

```
 * @author Carsten Weinholz
 *
 * @version $Revision: 1.8 $
 *
 * @since 6.0.0
 */
public class CmsDocumentXmlContentHispano extends A_CmsVfsDocument {



    /**
     * Creates a new instance of this lucene document factory.<p>
     *
     * @param name name of the documenttype
     */
    public CmsDocumentXmlContentHispano(String name) {

        super(name);
        System.out.println("ENTRA");
    }

    /**
     * Generates a new lucene document instance from contents of the given resource.<p>
     *
     * @see
org.opencms.search.documents.I_CmsDocumentFactory#newInstance(org.opencms.file.CmsObject,
org.opencms.search.A_CmsIndexResource, java.lang.String)
     */
    public Document newInstance(CmsObject cms, A_CmsIndexResource indexResource, String
language) throws CmsException {

        Document document = super.newInstance(cms,indexResource,language);

        StringBuffer meta = new StringBuffer(512);

        Field field;

        // Añadimos el valor de cada campo
        CmsResource resource = (CmsResource)indexResource.getData();
        String result = null;

        try {
            CmsFile file = CmsFile.upgrade(resource, cms);
            String absolutePath = cms.getSitePath(file);
            A_CmsXmlDocument xmlContent = CmsXmlContentFactory.unmarshal(cms, file);
            List locales = xmlContent.getLocales();
            if (locales.size() == 0) {
                locales = OpenCms.getLocaleManager().getDefaultLocales(cms,
absolutePath);
            }
            Locale locale = OpenCms.getLocaleManager().getBestMatchingLocale(
                CmsLocaleManager.getLocale(language),
                OpenCms.getLocaleManager().getDefaultLocales(cms, absolutePath),
                locales);

            List elements = xmlContent.getNames(locale);
            StringBuffer content = new StringBuffer();
            for (Iterator i = elements.iterator(); i.hasNext();) {
                I_CmsXmlContentValue value = xmlContent.getValue((String)i.next(),
locale);
                String plainText = value.getPlainText(cms);
                if (CmsStringUtil.isNotEmpty(plainText)) {
                    plainText = plainText.trim();
                    if (plainText.length() > 0) {
                        //Añadimos a los indices, marcando el nombre del campo con el
```

```
prefijo: xml-
                        field = new Field("xml-"+value.getName(), plainText,
Field.Store.YES, Field.Index.UN_TOKENIZED);
                        // title keyword field should not affect the boost factor
                        field.setBoost(0);
                        document.add(field);
                    }
                }
            }

            result = content.toString();
            // CmsHtmlExtractor extractor = new CmsHtmlExtractor();
            //rawContent = extractor.extractText(content.toString(), page.getEncoding());

        } catch (Exception e) {
            throw new CmsIndexException(
                Messages.get().container(Messages.LOG_CACHE_COSTS_TOO_HIGH_2,
resource.getRootPath()),
                e);
        }

        return document;

    }

    /**
     * Returns the raw text content of a given vfs resource of type
<code>CmsResourceTypeXmlContent</code>.<p>
     *
     * @see
org.opencms.search.documents.A_CmsVfsDocument#extractContent(org.opencms.file.CmsObject,
org.opencms.search.A_CmsIndexResource, java.lang.String)
     */
    public I_CmsExtractionResult extractContent(CmsObject cms, A_CmsIndexResource
indexResource, String language)
    throws CmsException {

        CmsResource resource = (CmsResource)indexResource.getData();
        String result = null;

        try {
            CmsFile file = CmsFile.upgrade(resource, cms);
            String absolutePath = cms.getSitePath(file);
            A_CmsXmlDocument xmlContent = CmsXmlContentFactory.unmarshal(cms, file);

            List locales = xmlContent.getLocales();
            if (locales.size() == 0) {
                locales = OpenCms.getLocaleManager().getDefaultLocales(cms,
absolutePath);
            }
            Locale locale = OpenCms.getLocaleManager().getBestMatchingLocale(
                CmsLocaleManager.getLocale(language),
                OpenCms.getLocaleManager().getDefaultLocales(cms, absolutePath),
                locales);

            List elements = xmlContent.getNames(locale);
            StringBuffer content = new StringBuffer();
            for (Iterator i = elements.iterator(); i.hasNext();) {
                I_CmsXmlContentValue value = xmlContent.getValue((String)i.next(),
locale);
                String plainText = value.getPlainText(cms);
                if (plainText != null) {
                    content.append(plainText);
                    content.append('\n');
                }
```

```
            }

            result = content.toString();
            // CmsHtmlExtractor extractor = new CmsHtmlExtractor();
            //rawContent = extractor.extractText(content.toString(), page.getEncoding());

        } catch (Exception e) {
            throw new CmsIndexException(
                Messages.get().container(Messages.LOG_CACHE_COSTS_TOO_HIGH_2,
resource.getRootPath()),
                e);
        }

        return new CmsExtractionResult(result);
    }

    /**
     * @see
org.opencms.search.documents.I_CmsDocumentFactory#getDocumentKeys(java.util.List,
java.util.List)
     */
    public List getDocumentKeys(List resourceTypes, List mimeTypes) throws CmsException {

        if (resourceTypes.contains("*")) {
            ArrayList allTypes = new ArrayList();
            for (Iterator i = OpenCms.getResourceManager().getResourceTypes().iterator();
i.hasNext();) {
                I_CmsResourceType resourceType = (I_CmsResourceType)i.next();
                if (resourceType instanceof CmsResourceTypeXmlContent
                    &&
((CmsResourceTypeXmlContent)resourceType).getConfiguration().containsKey(
                        CmsResourceTypeXmlContent.CONFIGURATION_SCHEMA)) {
                    allTypes.add(resourceType.getTypeName());
                }
            }
            resourceTypes = allTypes;
        }

        return super.getDocumentKeys(resourceTypes, mimeTypes);
    }
}
```

Una vez que tenemos creada nuestra clase, debemos compilarla y subirla a OpenCms. Para esto tenemos la opción de subir nuestro .class directamente, o generar un .jar y subirlo a la carpeta lib. Es aconsejable crear nuestro propio modulo o usar el mismo del frontend.

Llegado a este punto debemos indicarle a lucene que use nuestra nueva clase en lugar de la anterior, para ello nos vamos a editar el fichero $TOMCAT_HOME/webapps/opencms/WEB-INF/config/opencms-search.xml. Debemos buscar la entrada:

```
<documenttype>
       <name>xmlcontent</name>
       <class>org.opencms.search.documents.CmsDocumentXmlContent</class>
       <mimetypes/>
       <resourcetypes>
           <resourcetype>*</resourcetype>
       </resourcetypes>
</documenttype>
```

Y sustituirla por:

```
<documenttype>
       <name>xmlcontent</name>
       <class>org.opencmshispano.search.documents.CmsDocumentXmlContentHispano</class>
       <mimetypes/>
       <resourcetypes>
           <resourcetype>*</resourcetype>
       </resourcetypes>
</documenttype>
```

Una vez editado, reiniciamos nuestro servidor Tomcat, realizamos un rebuild de nuestro índice, y ya tendremos indexado, campo a campo, nuestros xmlContent.

6.- Comprobación

Una herramienta muy útil para trabajar con lucene es la denominada Luke-Lucene, que podemos encontrar en la página http://www.getopt.org/luke/. Con ella podremos ver el estado de nuestros índices, si tenemos almacenada la información que deseamos, y comprobar que todo es correcto. Dicha herramienta también está disponible en nuestro portal.

7.- Conclusión

Como ya he comentado en la motivación, un buscador en un portal es una de las características fundamentales. OpenCms, como hemos visto, nos facilita mucho la tarea de generar nuestro buscador. El potencial de lucene es mucho mayor de lo que aquí hemos mostrado, dándole solución a muchos de nuestros problemas con esta herramienta. Solo nos queda investigar, y aprender a sacarle el máximo provecho a esta potente herramienta.

Como siempre me gusta comentar, este tutorial no es ni mucho menos definitivo, sino una primera versión el cual, con vuestra ayuda, iré mejorando.

Autor: Sergio Raposo Vargas
administrador@opencmshispano.com