

Indicizzazione di Asset Custom

Claudio Umana, Application Architect at Sourcesense (Gruppo pro-Netics SpA)

Agenda

- Introduzione
- Cenni sul framework: indicizzare e ricercare
- Esempio di implementazione
- Ispezionare l'archivio indicizzato
- Estendere Liferay: Indexer Post Processor Hook

Introduzione

Un moderno CMS di classe enterprise non può non avere un sistema di indicizzazione dei contenuti che non sia:

- efficiente;
- versatile;
- espandibile;
- e che non permetta la ricerca fulltext.

Al tal fine *Liferay Portal* integra di default *Lucene* ed *Elastic Search* (LF>=7.0).

E' possibile usare anche SOLR.



Implementazione - 1

Panoramica

Liferay fornisce un Search Framework basato su *Apache Lucene* al fine di ricercare le informazioni funzionalmente valide di un proprio Asset Custom.

In linea di massima gli step principali da affrontare:

- Estendere la classe astratta `com.liferay.portal.kernel.search.BaseIndexer` che a sua volta implementa la `Indexer`;
- Implementare i metodi necessari per gestire l'indicizzazione;
- Definire classe indexer su `liferay-portlet.xml`;
- Annotare o modificare i metodi (cruciali per l'indicizzazione) nei service delle nostre entità;
- Ricercare le risorse custom.

In dettaglio...

Implementazione - 2

Metodi principali per l'indicizzazione della BaseIndexer

Affinché Lucene possa “manipolare” le nostre entità custom, è necessario implementare i metodi necessari per gestire l'indicizzazione (della **BaseIndexer**):

- `String[] getClassNames()`, array dei nomi dei modelli gestiti dall'indexer;
- `String getPortletId()`, identificativo univoco della portlet;
- `String getPortletId(SearchContext searchContext)`, identificativo univoco della portlet;
- `void doDelete(Object entity)`, per la cancellazione un'entità (per esempio che le eventuali entità collegate...);
- `Document doGetDocument(Object entity)`, si implementa la logica di mapping tra la l'entità custom e l'oggetto di tipo Document gestito da Lucene;
- `Summary doGetSummary(Document document, Locale locale, String snippet, PortletURL portletUrl)`, fornisce una rappresentazione sommaria di un oggetto Document restituito dalla ricerca su Lucene. Importante per gli *Asset Publisher* e della *Search Portlet*;
- `void doReindex(Object object)`, invocato dal portale quando è necessario il reindex. Alcuni esempi:
 - `List<MyEntity>` – una lista di entità da indicizzare; `MyEntity` – un'unica entità da indicizzare; `long[]` – un array di chiavi primarie delle entità da indicizzare; ...
- `void doReindex(String[] companyIds)`, invocato quando si effettua il reindex dal "Control Panel". In ingresso si fornisce l'elenco degli id interessati dal reindex;
- `void doReindex(String className, String classPK)`, invocato quando è necessario il reindex di una singola entità.

Implementazione - 3

Definizione classe indexer su liferay-portlet.xml

Nel nostro caso abbiamo bisogno di 3 riferimenti:

dal `liferay-portlet.xml` =>

```
...  
<portlet>  
...  
    <!-- Indexer -->  
    <indexer-class>it...HDCustomerIndexer</indexer-class>  
    <indexer-class>it...HDProductIndexer</indexer-class>  
    <indexer-class>it...HDProductReleaseIndexer</indexer-class>  
    ...  
</portlet>  
...
```

Questo è necessario per comunicare a Liferay la presenza di classi per la indicizzazione nella nostra portlet.

Implementazione - 4

Indicizzare - Annotation sui metodi service

Infine è necessario scatenare il processo di reindicizzazione quando l'entità viene **creata**, **modifica** o **eliminata**. Ci sono due modi per farlo:

- inserendo il codice corrispondente nel metodo di service;
- il secondo usando l'annotation `@Indexable` nei metodi interessanti, quelli che restituiscono l'entità da indicizzare.

Le classi interessate di service sono le `<MyEntity>LocalServiceImpl` e l'annotation prevede 2 differenti type di utilizzo:

- `IndexableType.REINDEX` – Si reindicizza l'entità su Lucene restituita dal metodo;
- `IndexableType.DELETE` – Si cancella l'entità restituita dal metodo dall'archivio degli indici.

Implementazione - 5

Indicizzare - Esempio annotation

Nei due esempi è mostrato l'uso:

```
@Indexable(type=IndexableType.REINDEX)
@Override
public MyEntity addMyEntity(MyEntity myEntity) throws SystemException {
    myEntity.setNew(true);
    return myEntityPersistence.update(myEntity);
}
```

```
@Indexable(type=IndexableType.DELETE)
@Override
public MyEntity deleteMyEntity(long myEntityId) throws PortalException, SystemException {
    return myEntityPersistence.remove(myEntityId);
}
```


Implementazione - 6

Ricerca - "Query Lucene"

Principalmente la ricerca delle entità fa uso delle "**Query Lucene**".

I metodi più usati:

- **SearchContext**, contenitore con i parametri di ricerca;
- **BooleanClause**, clausola di ricerca specifica su un campo;
- **IndexerRegistryUtil.nullSafeGetIndexer(String className)**, utility per recuperare l'oggetto Indexer associato al modello che si vuole cercare;
- **indexer.search(SearchContext searchContext)**, esegue la ricerca vera e propria. Restituisce un oggetto di tipo **com.liferay.portal.kernel.search.Hits**;
- **com.liferay.portal.kernel.search.Hits**, contiene tutti i dati relativi alla ricerca e al suo risultato. Importanti i metodi:
 - **getDocs()**, lista dei risultati (anche paginati), della query di ricerca. Si tratta di oggetti di tipo **Document**, usato da Lucene per rappresentare l'entità custom (si veda **doGetDocument()** dell'indexer);
 - **getLength()**, il totale dei risultati della ricerca anche nel caso di paginazione.

Implementazione - 7

Ricerca - "search"

Ecco un esempio:

```
public Hits search() throws SearchException
{
    SearchContext searchContext = new SearchContext();
    BooleanClause[] clauses = new BooleanClause[3];

    clauses[0] = BooleanClauseFactoryUtil.create(searchContext, "field1", "value 1", BooleanClauseOccurrence.MUST.toString());
    clauses[1] = BooleanClauseFactoryUtil.create(searchContext, "field2", "value 2", BooleanClauseOccurrence.SHOULD.toString());
    clauses[2] = BooleanClauseFactoryUtil.create(searchContext, "field3", "value 3", BooleanClauseOccurrence.MUST_NOT.toString());

    searchContext.setBooleanClauses(clauses);

    Indexer indexer = IndexerRegistryUtil.nullSafeGetIndexer(MyEntity.class);
    return indexer.search(searchContext); //tipo HITS
}
```

Implementazione - 8

Ricerca - SearchEngine

Altre modalità di ricerca sono riconducibili invece alla `SearchEngineUtil.search`.
Con le seguenti modalità di uso:

- `BooleanQuery`;
- `TermRangeQuery`;
- `TermQuery`.

Esempio 1 - *TermQuery*.

```
...  
TermQuery termQuery = TermQueryFactoryUtil.create(searchContext, "title", "liferay");  
try {  
    Hits hits = SearchEngineUtil.search(searchContext, termQuery);  
}  
catch (SearchException se) {  
    // handle search exception  
}  
...
```

Implementazione - 9

Esempio 2 - *TermRangeQuery*.

```
...  
TermRangeQuery termRangeQuery = TermRangeQueryFactoryUtil.create(searchContext,  
"modified", "201412040000000", "201412050000000", true, true);  
try {  
    Hits hits = SearchEngineUtil.search(searchContext, termRangeQuery);  
}  
catch (SearchException se) {  
    // handle search exception  
}  
...
```

Nota: i 2 'true' indicano se il range è con gli estremi compresi.

Nota: "modified" è il campo di ricerca di tipo data.

Implementazione - 10

Esempio 3 - Più ricco.

```
...
TermQuery termQuery1 = TermQueryFactoryUtil.create(searchContext, "title", "liferay");
TermQuery termQuery2 = TermQueryFactoryUtil.create(searchContext, "description", "lucene");
TermRangeQuery termRangeQuery = TermRangeQueryFactoryUtil.create(searchContext, "modified",
"201412040000000", "201412050000000", true, false);

BooleanQuery booleanQuery = BooleanQueryFactoryUtil.create(searchContext);

booleanQuery.add(termQuery1, BooleanClauseOccur. MUST);
booleanQuery.add(termQuery2, BooleanClauseOccur. MUST_NOT);
booleanQuery.add(termRangeQuery, BooleanClauseOccur. MUST);

try {
    Hits hits = SearchEngineUtil.search(searchContext, booleanQuery);
}
catch (SearchException se) {
    // handle search exception
}
...
```

Implementazione - 11

Oltre alle modalità indicate, è possibile usare anche la ***StringQuery***. Questa permette di usare la sintassi delle query di *Lucene*.

In ogni caso, alla fine è necessario avere i Document:

```
...
Hits hits = SearchEngineUtil.search(...);
Document[] docs = hits.getDocs();
...
List<Document> docs = hits.toList();
...
```

Esempio - 1

Esempio con i Product

Qui di seguito un esempio estratto per l'*HDPProductLocalServiceImpl*:

```
public List<HDPProduct> advancedSearch(long companyId, long groupId, String productTitle, String productDescription, boolean andSearch) {
    Hits hits = null;
    try {
        hits = getHits(companyId, groupId, productTitle, productDescription, andSearch);
    } catch (SystemException e) { ...}

    if (hits == null || hits.getLength() == 0)
        return null;

    List<HDPProduct> products = new ArrayList<HDPProduct>();
    for (int i = 0; i < hits.getLength(); i++) {
        Document doc = hits.doc(i);
        long productId = GetterUtil.getLong(doc.get(Field.ENTRY_CLASS_PK));
        try {
            HDPProduct product = fetchHDPProduct(productId);
            products.add(product);
        } catch (SystemException e) { ...}
    }
    return products;
}
```

Esempio - 2

Esempio con i Product

Dove la *getHits*:

```
protected Hits getHits(long companyId, long groupId, String productTitle, String productDescription, boolean andSearch) throws SystemException {
    SearchContext searchContext = new SearchContext();
    searchContext.setAndSearch(andSearch);
    searchContext.setCompanyId(companyId);
    long[] groupIds = { groupId };
    searchContext.setGroupIds(groupIds);

    BooleanQuery searchQuery = BooleanQueryFactoryUtil.create(searchContext);

    appendSearchTerm(Field.TITLE, productTitle, andSearch, searchQuery);
    appendSearchTerm(Field.DESCRPTION, productDescription, andSearch, searchQuery);

    Hits hits = null;
    try {
        hits = SearchEngineUtil.search(searchContext, searchQuery);
    } catch (SearchException e) { ... }
    return hits;
}
```


Esempio - 3

Esempio con i Product

Dove la *appendSearchTerm*:

```
private void appendSearchTerm(Stringfield, String value, boolean isAndSearch, BooleanQuery searchQuery) {
    if (isAndSearch) {
        searchQuery.addRequiredTerm(field, value, true);
    } else {
        try {
            searchQuery.addTerm(field, value, true);
        } catch (ParseException e) {...}
    }
}
```

la `view_products.jsp` che invocherà la ricerca:

```
<portlet:actionURL var="searchAdvancedURL" name="searchProducts" />

<auiform action="<%= searchAdvancedURL %>">
    <auiformselect name="searchType" label="Search Type" inlineLabel="true">
        <auiformoption value="<%= false %>" label="Any"/>
        <auiformoption value="<%= true %>" label="All"/>
    </auiformselect>
    <auiforminput name="productTitle"/>
    <auiforminput name="productDescription" />
    <auiformbutton type="submit" value="Search"/>
</auiform>
```

Esempio - 4

Esempio con i Product

Infine nella portlet, l'action di ricerca:

```
public void searchProducts(ActionRequest actionRequest, ActionResponse actionResponse) throws IOException, PortletException {
    ThemeDisplay themeDisplay = (ThemeDisplay) actionRequest.getAttribute(WebKeys.THEME_DISPLAY);

    String productTitle = ParamUtil.getString(actionRequest, "productTitle");
    String productDescription = ParamUtil.getString(actionRequest, "productDescription");
    String andSearch = ParamUtil.getString(actionRequest, "searchType");

    List<HDProduct> products = HDProductLocalServiceUtil.advancedSearch(themeDisplay.getCompanyId(), themeDisplay.getScopeGroupId(),
    productTitle, productDescription, new Boolean(andSearch));

    actionRequest.setAttribute("SEARCH_RESULT", products);
}
```

Esempio - 5

Ecco due ricerche sui Product...

base Customers Products

Search Type Any

Title
windows

Description
vista

Search

Add Product

Name	Title	Status	
windows XP	Windows XP	0 (Approved)	Actions
Windows 7	Windows seven	0 (Approved)	Actions
Windows 8	Windows 8	0 (Approved)	Actions
Windows 10	Windows 10	0 (Approved)	Actions

base Customers Products

Search Type All

Title
windows

Description
vista

Search

Add Product

Name	Title	Status	
windows XP	Windows XP	0 (Approved)	Actions

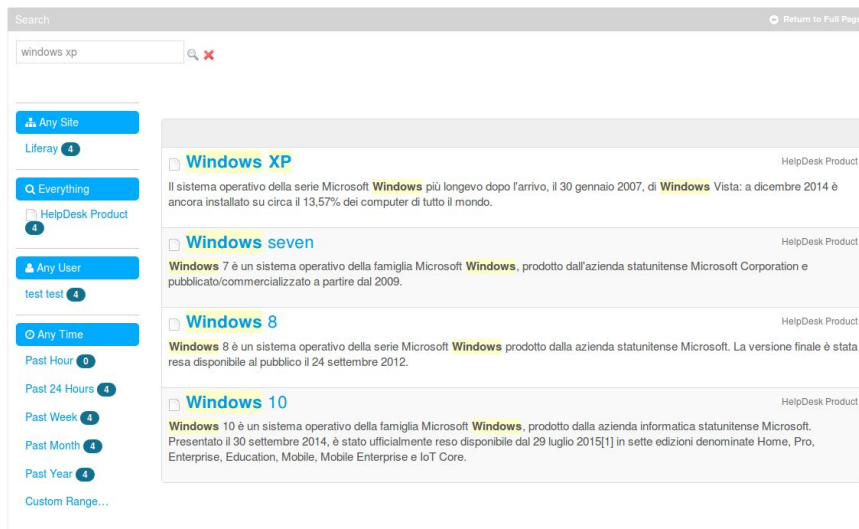
Faceted Search

E' noto che Liferay supporta la Faceted Search attraverso la "*Search Portlet*". Questo componente è configurabile:

- Configuration -> Display Settings -> Advanced -> Search Configuration e si usa un **JSON string** per le entità alla ricerca. In questa sede è necessario aggiungere le seguenti (in **neretto**):

```
{
  "displayStyle": "asset_entries",
  ...,
  "com.liferay.portlet.wiki.model.WikiPage",
  "it...model.HDPProduct",
  "it...model.HDPProductRelease",
  "it...model.HDCustomer"
  ...
}
```

Questo comunica alla portlet le entità interessate alla ricerca.



“Vedere” gli indici *Lucene* - 1

E' possibile ispezionare gli indici di *Lucene* generati e gestiti da Liferay.

Un tool java è **Luke** (<https://code.google.com/p/luke/>). Permette:

- navigare per numero di documento o per parole;
- vedere e copiare i documenti;
- definire dei rank sulla frequenza dei termini;
- fare ricerche e mostrare il risultato nei browser;
- analizzare i risultati delle ricerche;
- cancellare i documenti dall'indice;
- ricostruire i campi del documento, editare e reinserire gli indici;
- ottimizzare gli indici;
- ecc...

Vediamo un dettaglio dall'esempio di documento.

“Vedere” gli indici *Lucene* - 2

Prendiamo dalla nostra lista prodotti:

Add Product			
Name	Title	Status	
windows XP	Windows XP	0 (Approved)	Actions
Windows 7	Windows seven	0 (Approved)	Actions
Windows 8	Windows 8	0 (Approved)	Actions
Windows 10	Windows 10	0 (Approved)	Actions

il prodotto “Windows XP”:

Help Desk (Admin)

[←](#) Windows XP

[Edit](#) [Permissions](#) [Delete](#) [Status](#) Approved

ID

1

Name

windows XP

Title

Windows XP

Description

Il sistema operativo della serie Microsoft Windows più longevo dopo l'arrivo, il 30 gennaio 2007, di Windows Vista: a dicembre 2014 è ancora installato su circa il 13,57% dei computer di tutto il mondo.

Your Rating

Average (0 Votes)

☆☆☆☆☆☆☆☆☆☆

☆☆☆☆☆☆☆☆☆☆

“Vedere” gli indici *Lucene* - 3

Usando *Luke* per ispezionare gli indici generati da *Liferay*, si osservano i campi (definiti per la ricerca) funzionalmente validi: **description** e **title**.

The screenshot shows the Luke application interface. The top navigation bar includes tabs for Overview, Documents, Search, Files, and Plugins. The main window is divided into several sections:

- Browse by document number:** Shows document 29. Buttons include Add, Reconstruct & Edit, Delete, and More like this...
- Browse by term:** Allows searching by term. The term 'title' is entered. Buttons include First Term, Term: title, Decoded value, and Next Term.
- Browse documents with this term:** Shows 1 document. Buttons include Document: 1 of 1, First Doc, Next Doc, Show All Docs, and Delete All Docs.
- Delete specified list of documents:** Includes a text input and a Del List button.
- Example:** 0,12,45-90,17,128,30-32
- Doc #: 29** section with a table of fields and values.
- Flags:** I - Indexed (docs,freq,pos) T - Tokenized; S - Stored; V - Term Vector (offsets,pos) N - with Norms; L - Lazy; B - Binary; # - Numeric.

Field	IdfpTSvopNLB#	Norm	Value
companyId	Idfp-S---N---	0.0	20155
createDate	Idfp-S---N---	1.0	20151110101412
createDate	Id--TS-----#	---	1447150452000
description	IdfpTS---N---	0.15625	Il sistema operativo della serie Microsoft Windows più longevo dopo l'arrivo, il 30 gennaio 2007, di Windows Vista: a dicembre 20
entryClassN	Idfp-S---N---	0.0	it.smc.liferay.helpdesk.model.HDProduct
entryClassP	Idfp-S---N---	0.0	1
expirationD	Idfp-S---N---	1.0	2922789940817071255
expirationD	Id--TS-----#	---	9223372036854775807
groupid	Idfp-S---N---	0.0	20182
localized tit	Idfp-S---N---	1.0	windows xp
localized tit	Idfp-S---N---	1.0	windows xp
ratings	Idfp-S---N---	1.0	0.0
ratings sort	Id--TS-----#	---	0.0
roleId	Idfp-S---N---	1.0	20164
scopeGroup	Idfp-S---N---	0.0	20182
stagingGrou	Idfp-S---N---	1.0	false
status	Idfp-S---N---	1.0	0
title	IdfpTS---N---	0.625	Windows XP
title sortab	Idfp-S---N---	1.0	windows xp
uid	Idfp-S---N---	1.0	1 WAR help deskportlet PORTLET 1
userid	Idfp-S---N---	0.0	20199
userName	Idfp-S---N---	1.0	test test
viewCount	Idfp-S---N---	1.0	0
viewCount	Id--TS-----#	---	0
visible	Idfp-S---N---	1.0	true

Selected field: TV Show Set norm Save Copy text to Clipboard: Selected fields Complete document

Estendere l'indicizzazione - 1

Lo strumento fornito da Liferay è il *Indexer Post Processor Hook*.

Vedremo:

- come usare l'hook
- 2 esempi: *OpenId* per *User* e il *TreePath* per il *JournalArticle*.

Il primo intervento è nel `docroot/WEB-INF/liferay-hook.xml`:

```
<indexer-post-processor>
  <indexer-class-name>com.liferay.portal.model. User</indexer-class-name>
  <indexer-post-processor-impl>it...hook.indexer. UserIndexerExtensionPostProcessor</indexer-post-processor-impl>
</indexer-post-processor>
<indexer-post-processor>
  <indexer-class-name>com.liferay.portlet.journal.model. JournalArticle</indexer-class-name>      <indexer-post-processor-impl>it...hook.indexer. JournalArticleIndexerExtensionPostProcessor</indexer-post-processor-impl>
</indexer-post-processor>
```

Dove:

- **indexer-class-name**, è il model entity da "estendere";
- **indexer-post-processor-impl**, la classe che ne implementa l'interfaccia.

Estendere l'indicizzazione - 2

Per il *JournalArticle*:

```
public class JournalArticleIndexerExtensionPostProcessor extends BaseIndexerPostProcessor {
    ...
    public void postProcessDocument(Document document, Object object) throws Exception {
        JournalArticle journalArticleEntity = (JournalArticle) object;
        String indexerJournalArticleTreePath = "";
        try {
            indexerJournalArticleTreePath=journalArticleEntity.getTreePath();
        } catch (Exception e) {...}
        if(indexerJournalArticleTreePath.length() > 0 document.addText("treePath", indexerJournalArticleTreePath);
        }
        ...
    public void postProcessSearchQuery(BooleanQuery searchquery, SearchContext searchcontext) throws Exception {
        addSearchTerm(searchquery, searchcontext, "treePath", false);
    }
    ...
    protected void addSearchTerm(BooleanQuery searchQuery, SearchContext searchContext, String field, boolean like) throws Exception {
        ...
        if (searchContext.isAndSearch()) { searchQuery.addRequiredTerm(field, value, like);
        } else { searchQuery.addTerm(field, value, like); }
    }
    ...
}
```

dove `addRequiredTerm` è una clause in **MUST**, la `addTerm` è una **SHOULD**.

Estendere l'indicizzazione - 3

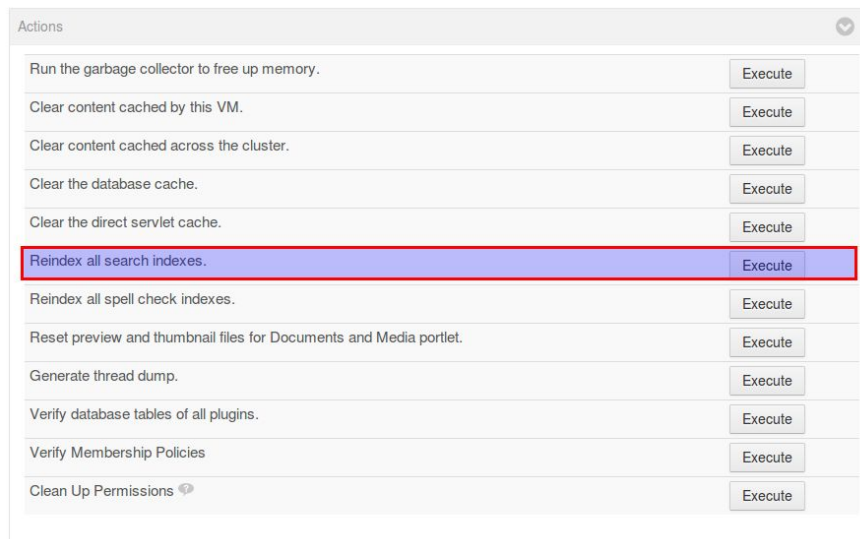
Per lo *User*:

```
public class UserIndexerExtensionPostProcessor extends BaseIndexerPostProcessor {
    ...
    public void postProcessDocument(Document document, Object object) throws Exception {
        User userEntity = (User) object;
        String indexerUserOpenId = "";
        try {
            indexerUserOpenId = userEntity.getOpenId();
        } catch (Exception e) { ... }
        if (indexerUserOpenId.length() > 0) document.addText(openId", indexerUserOpenId);
    }
    ...
    public void postProcessSearchQuery(BooleanQuery searchquery, SearchContext searchcontext) throws Exception {
        ...
        addSearchTerm(searchquery, searchcontext, openId", false);
    }
    ...
    protected void addSearchTerm(BooleanQuery searchQuery, SearchContext searchContext, String field, boolean like) throws Exception {
        ...
        if (searchContext.isAndSearch()) { searchQuery.addRequiredTerm(field, value, like);
        } else {searchQuery.addTerm(field, value, like);}
    }
    ...
}
```

dove **addRequiredTerm** è una clause in **MUST**, la **addTerm** è una **SHOULD**.

Estendere l'indicizzazione - 4

Si installa l'hook e si **reindicizza!**



Grazie per l'attenzione

claudio.umana@i3solevo.it
claudio.umana@sourcesense.it



sourcesense
making sense of open source

