Indicizzazione entità custom con Lucene

Come possiamo utilizzare Lucene per la ricerca delle nostre entità custom?

Liferay utilizza il motore di indicizzazione Lucene per eseguire le ricerche su molte delle sue entità di portale. In questo articolo vedremo quali sono i passi per implementare ricerche di questo tipo anche sulle entità delle nostre portlet custom.

Implementare l'indexer

La prima cosa da fare è descrivere come tradurre la nostra entità custom in modo che Lucene possa gestirla: per fare questo dobbiamo implementare un nostro "indexer". L'indexer sarà una classe che deve estendere la classe astratta com.liferay.portal.kernel.search.BaseIndexer

I metodi principali che dobbiamo implementare:

- String[] getClassNames() – fornisce un array dei nomi dei modelli che il nostro indexer gestisce
- String getPortletId() – fornisce l'identificativo univoco della nostra portlet
- String getPortletId(SearchContext searchContext) – analogo al metodo precedente
- void doDelete(Object entity) – le operazioni da eseguire sugli indici Lucene quando viene eliminata un'entità (ad es. eliminare anche le entità correlate)
- Document doGetDocument(Object entity) – il metodo principale, in cui dobbiamo implementare la logica di mapping tra la nostra entità custom e l'oggetto di tipo Document gestito direttamente da Lucene e salvato sugli

indici

- Summary doGetSummary(Document document, Locale locale, String snippet, PortletURL portletUrl) – restituisce una rappresentazione sommaria di un oggetto Document proveniente da una ricerca su Lucene. Sebbene questo metodo non sia strettamente necessario per l'utilizzo degli indici di lucene, viene utilizzato in molte portlet del portale, quali Asset Publisher o Ricerca, quindi consiglio di implementarlo almeno in bozza
- void doReindex(Object object) – invocato dal portale tutte le volte che è necessario il reindex. Il parametro in ingresso generico permette di gestire diversi casi, ad esempio:
    - List<[MyEntity]> – una lista di entità da indicizzare
    - [MyEntity] – un'unica entità da indicizzare
    - long[] – un array di chiavi primarie delle entità da indicizzare
    - etc.
- void doReindex(String[] companyIds) – invocato quando si effettua il reindex da pannello di controllo (Vedi "Reindex completo entità della portlet" per dettagli). In ingresso vengono ricevuti gli id di tutte le company presenti nel portale
- void doReindex(String className, String classPK) – invocato quando è necessario il reindex di una singola entità, conosciuti nome della classe di modello (className) e chiave univoca (classPK)

Come spesso accade nel mondo Liferay ci sono ottimi esempi di implementazione all'interno del codice stesso del portale. In questo caso consiglio di studiare la classe com.liferay.portlet.usersadmin.util.UserIndexer che è quella utilizzata dal portale per indicizzare gli utenti e contiene ottimi spunti e punti di riferimento implementativi per tutti i metodi descritti sopra

Definizione classe indexer su liferay-portlet.xml

Oltre ad implementare l'indexer dobbiamo comunicare a liferay che la nostra portlet utilizzare un indexer. Per farlo dobbiamo inserire nel liferay-portlet.xml del nostro progetto il tag <indexer-class> con l'indicazione del nome completo (package + nome classe) del nostro indexer

```
<portlet>
    <portlet-name>test-lucene1</portlet-name>
    <icon>/icon.png</icon>
    <indexer-class>it.marcorosetti.test.search.MyEntityIndexer</indexer-class>
    <header-portlet-css>/css/main.css</header-portlet-css>
    <footer-portlet-javascript>
```

Annotation su metodi del Service

Abbiamo quasi finito, ci manca solo di invocare il processo di reindicizzazione ogni volta che una nostra entità viene creata, modificata o eliminata. Tra i vari modi che ci sono per invocare la reindicizzazione quello che consiglio è l'utilizzo dell'annotation @Indexable. Questa annotation può essere messa sui metodi dei services (tipicamente nella classe [MyEntity]LocalServiceImpl) che hanno come parametro di ritorno l'entità che vogliamo reindicizzare: in questo modo, ogni volta che un metodo annotato ritorna un'entità, Liferay si preoccupa di intercettare l'entità e di eseguirne il reindex sugli indici di Lucene.
L'annotation prevede il parametro "type", obbligatorio, che può avere 2 valori e ne configura il comportamento:

- IndexableType.REINDEX – L'entità ritornata viene reindicizzata sugli indici di Lucene

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

    return myEntityPersistence.update(myEntity);
}
```

- IndexableType.DELETE – L'entità ritornata viene rimossa

dagli indici di Lucene

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

Ricerca di un'entità

La ricerca di entità avviene utilizzando principalmente le query Lucene. Si tratta di un argomento molto vasto e complesso che cercherò di affrontare in un articolo separato.

La metodologia "base" per poter eseguire una ricerca si fonda su questi oggetti e metodi

- SearchContext – contenitore di tutti i parametri di ricerca
- BooleanClause – rappresentazione di una clausola di ricerca specifica su un campo per un particolare valore
- IndexerRegistryUtil.nullSafeGetIndexer(String className) – utility per recuperare l'oggetto Indexer associato al modello che si vuole cercare
- indexer.search(SearchContext searchContext) – Metodo che esegue la ricerca vera e propria. Ritorna un oggetto di tipo com.liferay.portal.kernel.search.Hits.
- com.liferay.portal.kernel.search.Hits – Oggetto che contiene tutti i dati relativi alla ricerca e al suo risultato. Da segnalare in particolare i metodi
    - getDocs() – La lista dei risultati, eventualmente paginati, della query di ricerca. Si tratta di oggetti di tipo Document, l'oggetto specifico che utilizza Lucene come rappresentazione della nostra entità (si veda il metodo doGetDocument() dell'indexer per dettagli)
    - getLength() – Il totale dei risultati della ricerca. Da notare che, in caso di paginazione, il metodo getDocs() tornerà una sola pagina di

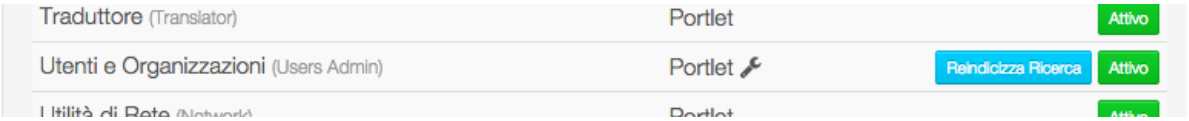
elementi, mentre getLength() tornerà il numero totale degli elementi della ricerca

```
public Hits search() throws SearchException
{
    SearchContext searchContext = new SearchContext();

    BooleanClause[] clauses = new BooleanClause[3];

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

    searchContext.setBooleanClauses(clauses);

    Indexer indexer = IndexerRegistryUtil.nullSafeGetIndexer(MyEntity.class);
    Hits hits = indexer.search(searchContext);
    return hits;
}
```

Reindex completo entità della portlet

Può capitare di dover reindicizzare in una volta sola tutte le entità di una portlet (ad esempio dopo una modifica batch su database). Per farlo Liferay mette a disposizione una funzionalità che lancia il reindex di una portlet su tutte le comunità configurate nel portale.

Per farlo, nel Pannello di Controllo, scheda App, cercare la portlet che si vuole reindicizzare e cliccare su Reindicizza Ricerca



Riferimenti e approfondimenti

- http://lucene.apache.org/
- https://code.google.com/p/luke/ – Tool per l'apertura e la navigazione degli indici di Lucene
- http://blog.d-vel.com/web/blog/home/-/blogs/inidicizzazione-e-ricerca-in-liferay
- http://www.scalsys.com/blog/implementing-lucene-search-in-liferay-custom-portlet/