Lucene: sincronizzarsi con l'indicizzazione

Come possiamo essere sicuri che l'indicizzazione di Lucene sia terminata?

Lucene è un motore di indicizzazione utilizzato da Liferay per la ricerca delle proprie entità e può essere facilmente implementato anche per le entità custom delle proprie portlet. Oltre agli indiscutibili vantaggi in termini di possibilità di ricerca e di velocità possiamo però incorrere in alcuni piccoli problemi o comportamenti inaspettati.

Uno di questi deriva dal fatto che il processo di indicizzazione vero e proprio viene eseguito in modo asincrono (cioè su un thread separato) rispetto al thread principale da cui parte. Questa scelta, unita ad un processo di creazione/aggiornamento dell'entità sugli indici che potrebbe richiedere molto tempo, può condurre a risultati inattesi nelle ricerche eseguite nei momenti successivi.

La soluzione a questo problema richiede non più di due righe di codice ma è utile avere una visione un po' più ampia dell'intero meccanismo di reindicizzazione per capirne meglio il funzionamento.

Un esempio per chiarire il problema

Abbiamo un'entità che, tra le altre, ha una variabile che ne identifica il colore che può assumere differenti valori(giallo, rosso, verde, etc.). Abbiamo una visualizzazione a lista che ci permette di filtrare le diverse entità per singolo colore e per fare questo esegue una query sugli indici Lucene.

Supponiamo di avere il filtro impostato su “Giallo” e di modificare una delle entità una cambiandone il colore. Al salvataggio viene avviato da Liferay il processo di indicizzazione Lucene dell’entità ma non si attende che questo abbia effettivamente terminato l’indicizzazione prima di ritornare il controllo ai livelli superiori. In altre parole l’utente che ha fatto il salvataggio può ricevere la notifica di operazione completata prima che l’aggiornamento degli indici sia terminato. Se in questa situazione viene ricaricata la lista con il filtro a “Giallo”, l’entità incriminata continuerà a comparire anche se il salvataggio è andato a buon fine.

Il Message Bus per ottenere processi asincroni

L’asincronicità dell’indicizzazione è ottenuta da Liferay sfruttando i servizi del Message Bus. Questi servizi permettono, tra le altre cose, di inviare messaggi a destinazioni (chiamate code) senza preoccuparsi né di chi né di quando verranno ricevuti e processati.

Nel nostro caso particolare, in seguito al salvataggio, Liferay invia un messaggio sul Message Bus contenente tutti i dati utili alla reindicizzazione su una coda specifica senza attendere una risposta. Da parte di Lucene ci sarà un processo incaricato di verificare la presenza di messaggi di reindicizzazione sulla coda ed eventualmente processarli, ma su un thread separato da quello che li ha generati

Forzare l’attesa di risposta ad un messaggio

Esiste però un meccanismo per forzare Liferay ad attendere una risposta ogni volta che invia un messaggio sul Message Bus e questo si ottiene inserendo questo codice in un punto

precedente all’operazione di indicizzazione

ProxyModeThreadLocal.setForceSync(true);
Può essere una buona idea (ma va valutata a seconda dei casi specifici) inserire il codice prima di ogni Action della nostra portlet. Per fare questo è sufficiente fare l’override del metodo callActionMethod nella nostra classe che estende MVCPortlet in questo modo

```
/**
 * Portlet implementation class TestPortlet1
 */
public class TestPortlet1 extends MVCPortlet {


    @Override
    protected boolean callActionMethod(
        ActionRequest actionRequest, ActionResponse actionResponse)
        throws PortletException {

        boolean forceSync = ProxyModeThreadLocal.isForceSync();

        //marcorosetti - Ensure all thread is synchronous (lucene reindex too)  to avoid problem
        //when reloading list (before reindex process has finished)
        ProxyModeThreadLocal.setForceSync(true);
        boolean result = super.callActionMethod(actionRequest, actionResponse);

        ProxyModeThreadLocal.setForceSync(forceSync);
        return result;
    }

```

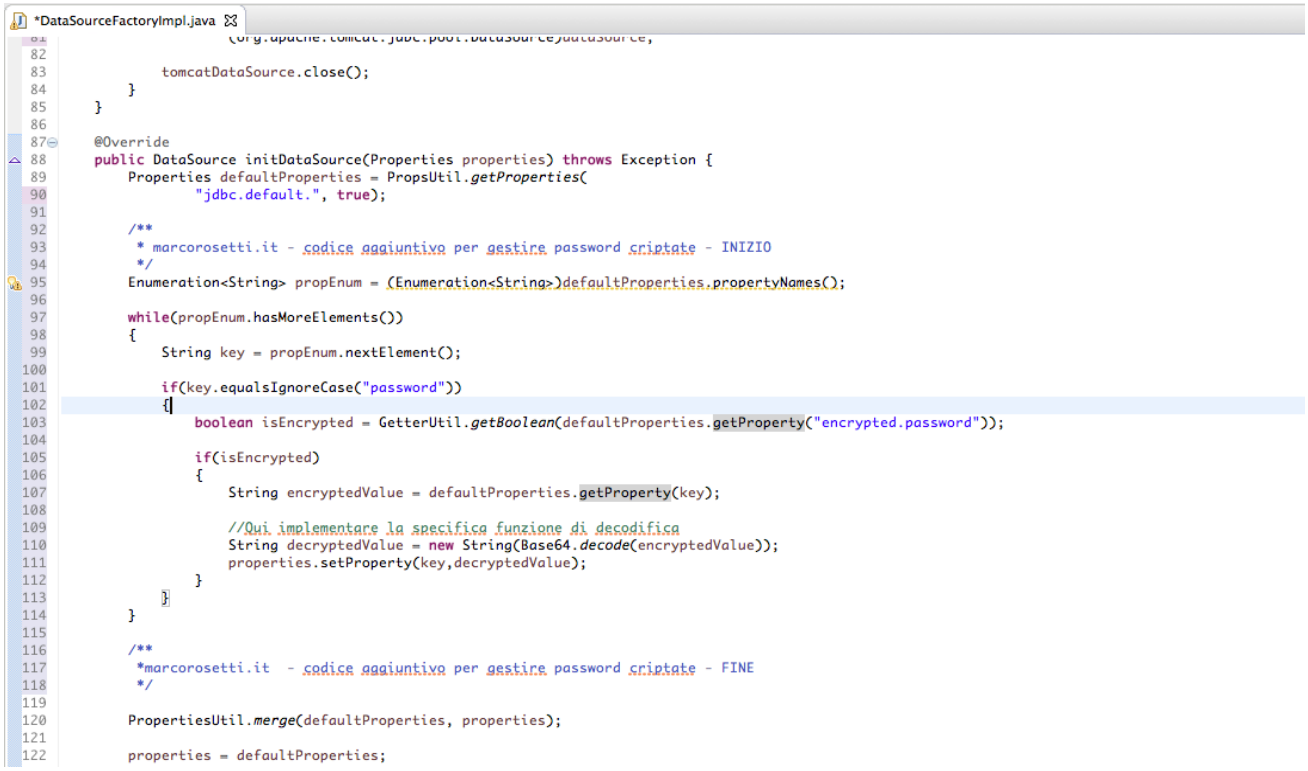
Password di accesso al database criptata

Come rendere più sicura la password di

connessione al database?

Le credenziali di accesso al database vengono salvate, di default, in chiaro all'interno di un file di properties (portal-ext.properties o portal-setup-wizard.properties a seconda dei casi). Se vogliamo rendere più sicuro questo meccanismo possiamo fare una semplice modifica al codice sorgente di Liferay per poter gestire anche valori criptati.

La classe che dobbiamo modificare è com.liferay.portal.dao.jdb.DataSourceFactoryImpl, in particolare il metodo initDataSource(Properties properties):



Vediamo le operazioni importanti del codice che abbiamo inserito

1. Scorriamo tutte le proprietà impostate fino a trovare la proprietà password
2. Se è stato impostato a TRUE il valore di jdbc.default.encrypted.password allora consideriamo il valore della password come criptato e lo decriptiamo

prima di sovrascriverlo nelle properties

Una volta modificato il codice dobbiamo rigenerare le librerie di liferay utilzzando i task ant messi a disposizione insieme ai sorgenti: possiamo utilizzare compile e poi andare a sostituire la classe generata all'interno del file **portal-impl.jar** nella cartella **tomcat-7.0.42\webapps\ROOT\WEB-INF\lib** oppure rigenerare l'intero file con il task jar.

Alcune considerazioni

- Nell'esempio riportato l'algoritmo di cript-decrypt è un semplice Base64 hashing che serve solo come "placeholder". Possiamo ovviamente complicare come vogliamo l'algoritmo a seconda delle nostre esigenze
- L'approccio applicato alla proprietà "password" si può applicare a qualsiasi proprietà si intenda criptare
- Se non facciamo altre modifiche ai sorgenti potrebbero venire stampati nei log dei messaggi a livello WARN del tipo "*Property encrypted.password is not a valid…*". Possiamo tranquillamente ignorare questi messaggi
- L'esempio vale per Liferay versione 6.2
- La sicurezza di questo approccio potrebbe essere aumentata andando ad offuscare il codice della classe
- Un approccio di sicurezza più architetturale consiste nell'utilizzare datasource esterni e iniettati tramite nome JNDI. In questo modo la gestione della sicurezza delle credenziali di accesso è demandata all'application server