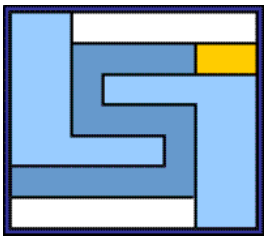

Item 10
Lucene Full Text Search

Diseño y Pruebas

Grado de Ingeniería del

Software Curso 3

César
García
Pascual

Fecha: 08 de mayo de 2018

1. Introduction

Lucene's full text search engine allows to index a database based on the fields that has the information required and is optimized to compute the intersection, union, etc. The SQL LIKE operator can be extremely inefficient. Indexing and using Lucene makes the search faster and lighter for the database.

2. Modified files.

- MODIFY: Article.java on domain package.
- MODIFY: pom.xml added hibernate-search dependency
- ADD: LuceneService on services package.
- ADD: LuceneController on controllers package.
- ADD: search.jsp and list.jsp with their corresponding messages y tiles for localization in lucene folder on the views folder.
- MODIFY: Added lucene tiles location on tiles.xml on main/resources/spring/config.
- MODIFY: Added lucene messages location on i18n-l10n.xml on main/resources/spring/config.
- MODIFY: Added URL permissions on tiles.xml on main/resources/spring/config.

3. Scripts.

- **pom.xml**

```
<dependencies>

    <!--  A+ -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-search</artifactId>
        <version>5.0.1.Final</version>
    </dependency>
```

This is needed for implementing the full text search. I used this version because newer and older ones threw me problems when running PopulateDatabes.java.

- **Article.java**

```
@Indexed
public class Article extends DomainEntity {


    //Attributes----------------

    private String title;

    private String summary;

    private String body;

    private Collection<String> picturess;

    private Boolean finalMode;

    private Boolean inappropriate;

    private Date publicationMoment;

    @Field
    @NotBlank
    @NotNull
    @SafeHtml(whitelistType=WhiteListType.NONE)
    public String getTitle() {
        return title;
    }
```

Added annotation @Indexed before the header of the class and @Field to the string in whose search the keyword (title, summary and body).

- **LuceneService.java**

```
package services;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.PersistenceContextType;
import javax.persistence.Query;

import org.hibernate.search.jpa.FullTextEntityManager;
import org.hibernate.search.jpa.Search;
import org.hibernate.search.query.dsl.QueryBuilder;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import domain.Article;

@Service
@Transactional
public class LuceneService {

    //Inits ----------------------------------------------------------------------------------------------------------------------
    // Logger
    @SuppressWarnings("unused")
    private static Logger LOG = LoggerFactory.getLogger(LuceneService.class);

    //JPA Persistence Unit
    @PersistenceContext(type = PersistenceContextType.EXTENDED, name = "AcmeNewspaper")
    private EntityManager em;

    //Hibernate Full Text Entity Manager
    private FullTextEntityManager ftem;




    //Search Method --------------------------------------------------------------------------------------------------------------
    @SuppressWarnings("unchecked")
    @Transactional
    public List<Article> search(String searchString) {

        try {
            updateFullTextIndex();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }


        // Create a Query Builder
        QueryBuilder qb = getFullTextEntityManager().getSearchFactory().buildQueryBuilder().forEntity(Article.class).get();


        // Create a Lucene Full Text Query
        org.apache.lucene.search.Query luceneQuery = qb.bool()
            .must(qb.keyword().onFields("title", "body", "summary").matching(searchString).createQuery()).createQuery();


        Query fullTextQuery = getFullTextEntityManager().createFullTextQuery(luceneQuery, Article.class);


        System.out.println(getFullTextEntityManager().createFullTextQuery(luceneQuery, Article.class).toString());
        // Run Query and print out results to console
        List<Article> result = fullTextQuery.getResultList();




        return result;
    }

    //Auxiliary Methods ----------------------------------------------------------------------------------------------------------
    public void updateFullTextIndex() throws Exception {
        getFullTextEntityManager().createIndexer().startAndWait();
    }

    protected FullTextEntityManager getFullTextEntityManager() {
        if (ftem == null)
            ftem = Search.getFullTextEntityManager(em);
        return ftem;
    }


}
```

The EntityManager injection has to be obtained from the persistence context. The name of the bean is indicated at /main/resources/META-INF/persistence.xml. It basically

creates a query with the QueryBuilder searching for the keywords on the fields indicated (title, body and summary) and then gets the result using the method ".getResultList()".

- **LuceneController.java**

```
import services.LuceneService;
import controllers.AbstractController;
import domain.Article;

@Controller
@RequestMapping("/lucene")
public class LuceneController extends AbstractController {

    @Autowired
    private LuceneService   luceneService;

    //Constructor
    public LuceneController() {
        super();
    }

    @RequestMapping("/search")
    public ModelAndView search(){
        ModelAndView res;
        try{
            res = new ModelAndView("lucene/search");
            res.addObject("requestUri", "lucene/search.do");
        } catch(Throwable oops) {
            res = new ModelAndView("redirect:/");
        }
        return res;
    }

    @RequestMapping("/list")
    public ModelAndView list(@RequestParam(required=true)String keyword){
        ModelAndView res;
        try{
            List<Article> articles = luceneService.search(keyword);
            res = new ModelAndView("lucene/list");
            res.addObject("articles", articles);
            res.addObject("requestUri", "lucene/list.do");
        } catch(Throwable oops) {
            res = new ModelAndView("redirect:search.do");
        }
        return res;
    }

}
```

Created a simple controller with 2 get methods, listing and searching. The list can requires a string which is the keyword to be searched.

- **search.jsp**

```jsp
<%@page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<%@taglib prefix="jstl" uri="http://java.sun.com/jsp/jstl/core"%>
<%@taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@taglib prefix="security" uri="http://www.springframework.org/security/tags"%>
<%@taglib prefix="display" uri="http://displaytag.sf.net"%>
<%@taglib prefix="lib" tagdir="/WEB-INF/tags/myTagLib" %>
<%@taglib prefix="fmt" uri = "http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>


<form action="lucene/list.do">
  <div class="input-group col-sm-8 col-sm-offset-2 col-xs-10 col-xs-offset-1" style="margin-bottom:15px">
    <input type="text" name="keyword" class="form-control" placeholder="<spring:message code="search" />" value="${keyword}">
    <div class="input-group-btn">
      <button class="btn btn-default" type="submit">
        <i class="glyphicon glyphicon-search"> </i>
      </button>
    </div>
  </div>
</form>
```

A simple view with a text input and a button with action to lucene/list.

- **list.jsp**

```jsp
<%@taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@taglib prefix="security" uri="http://www.springframework.org/security/tags"%>
<%@taglib prefix="display" uri="http://displaytag.sf.net"%>
<%@taglib prefix="lib" tagdir="/WEB-INF/tags/myTagLib" %>
<%@taglib prefix="fmt" uri = "http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>


<form action="lucene/list.do">
  <div class="input-group col-sm-8 col-sm-offset-2 col-xs-10 col-xs-offset-1" style="margin-bottom:15px">
    <input type="text" name="keyword" class="form-control" placeholder="Search" value="${keyword}">
    <div class="input-group-btn">
      <button class="btn btn-default" type="submit">
        <i class="glyphicon glyphicon-search"> </i>
      </button>
    </div>
  </div>
</form>


<div class="col-sm-10 col-sm-offset-1 well">
    <display:table pagesize="5" class="displaytag" keepStatus="true" name="articles" requestURI="${requestUri}" id="row2">
        <display:setProperty name="paging.banner.onepage" value=""/>
        <display:setProperty name="paging.banner.placement" value="bottom"/>
        <display:setProperty name="paging.banner.all_items_found" value=""/>
        <display:setProperty name="paging.banner.one_item_found" value=""/>
        <display:setProperty name="paging.banner.no_items_found" value=""/>

    <jstl:set var='model' value='article' scope='request'/>

    <lib:column name="title"></lib:column>
    <lib:column name="publicationMoment" format="{0,date,dd/MM/yyyy}"></lib:column>
    <lib:column name="summary" value="${fn:substring(row2.summary,0,30)}..."/>

    <lib:column name="newspaper" link="newspaper/display.do?newspaperId=${row2.newspaper.id}" linkName="${row2.newspaper.title}"/>

    <spring:message code="article.display" var="display_header"/>
    <display:column title="${display_header}">
        <jstl:if test="${not row2.newspaper.isPrivate or articlesMap[row2]}">
            <a href="article/display.do?articleId=${row2.id}">${display_header}</a>
        </jstl:if>
    </display:column>

</display:table>
</div>
```

A view with the same search input and button but with a table with the results if found any.