Informe Técnico
marzo, 2015

Implementación de sistemas de búsqueda con Solr

Juan Luis Boya García

Departamento de Informática y Automática
Universidad de Salamanca
(http://diaweb.usal.es)

Resumen

Solr es una plataforma de búsqueda de código abierto basada en Apache Lucene. Funciona como un servidor HTTP y se integra con otras plataformas a través de servicios web.

El propósito de este informe es describir algunas de las características fundamentales de esta herramienta de cara a la implementación de sistemas de búsqueda dentro utilizables por un usuario humano en un sitio web.

Índice

1. Instalación

Desde la página web de Solr (http://lucene.apache.org/solr/), se puede descargar el fichero comprimido `solr-5.0.0.tgz`.

Para una instalación básica basta con descomprmir este archivo comprimido en un directorio.

Es necesario tener instalada una máquina virtual de Java, incluyendo los paquetes de desarrollo (JDK).

2. El servidor Solr

En las plataformas UNIX se puede iniciar el servidor ejecutando la siguiente orden:

```
bin/solr start
```

Alternativamente, Solr se puede iniciar en Windows con la siguiente orden:

```
bin\solr.cmd start
```

Por defecto se lanza una instancia en http://localhost:8983/solr/.

3. Creación de un nuevo core

Solr denomina *core* a cada instancia de su sistema de búsqueda.

Cada *core* funciona de forma independiente y tiene sus propios archivos de configuración, aunque desde la misma interfaz se pueden administrar varios cores.

4. Definición del esquema

El fichero `schema.xml`, dentro de la configuración de un *core*, define los tipos de dato que manejará el servidor Solr.

Los tipos básicos están basados en clases Java de Solr. Por ejemplo:

```
<fieldType name="tdouble"
           class="solr.TrieDoubleField"
           precisionStep="8"
           positionIncrementGap="0"/>
```

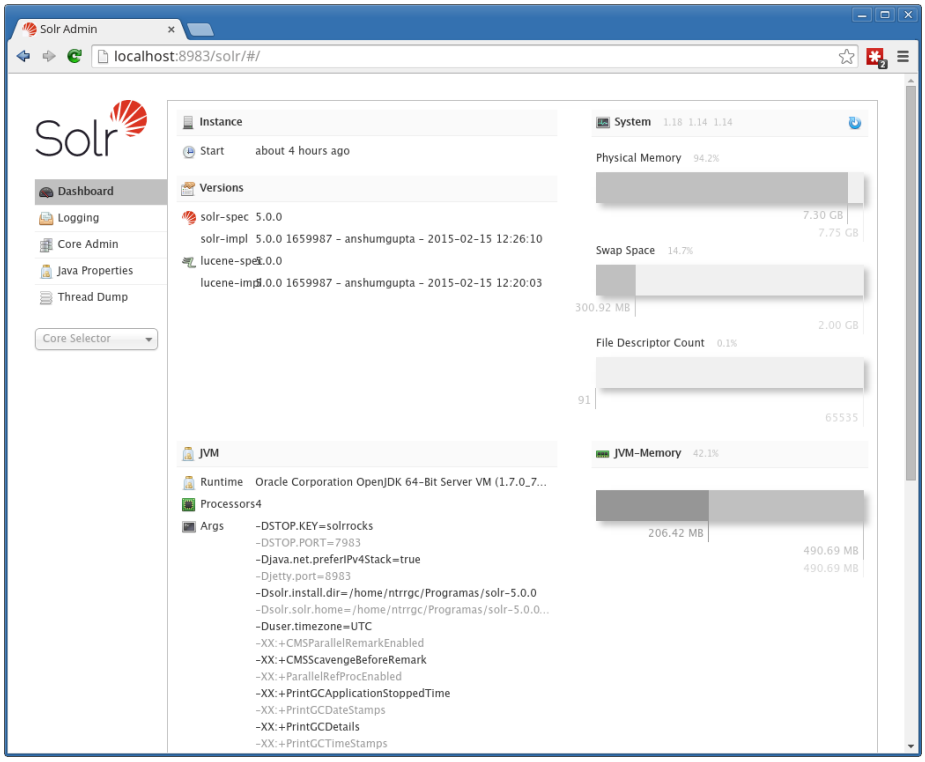

Figura 1: Interfaz web de Solr

4.1. Tipos de datos simples

Estos son algunos de los tipos más comunes:

- `int`: Número entero.
- `float`: Número decimal.
- `date`: Fecha y hora. Sólo acepta fechas UTC en formato ISO 8601. La letra `Z` al final de una fecha indica que esa fecha es UTC y es obligatoria.
- `boolean`: Admite los valores `true` y `false`.
- `string`: Admite cualquier longitud de texto.

4.2. Tipos de datos complejos

La característica más distintiva de Solr es la posibilidad de realizar búsquedas en texto completo de forma eficiente.

Para poder hacer búsquedas de forma eficiente en campos de texto es necesario que éstos sean analizados en tiempo de indexación.

Un tipo de dato de Solr puede incluir elementos `<analyzer/>`. Se especificarán dos, uno para el tratamiento de términos de búsqueda (`type="query"`) y otro para el tratamiento de documentos en tiempo de indexación (`type="index"`).

En cada uno se especificará una clase `tokenizer` que separará las palabras del documento y cero o más clases `filter` que recibirán estas secuencias de cadenas y devolverán otras con modificaciones.

Habitualmente se usan los mismos filtros en ambos analizadores.

4.3. Tokenizers

Según el tipo de texto a tratar un *tokenizer* puede ser más útil que otro. Estos son algunos de ellos:

- `solr.KeywordTokenizerFactory`: Devuelve un único *token* con toda la cadena.
- `solr.WhitespaceTokenizerFactory`: Los *tokens* están separados por espacios y saltos de línea.
- `solr.StandardTokenizerFactory`: Utiliza las reglas de separación de palabras definidas por Unicode en el anexo UAX#29 . Los símbolos, como punto o coma, también cortan palabras, salvo en algunas excepciones (ej. números de versiones con guiones).
- `solr.UAX29URLEmailTokenizerFactory`: Extiende `solr.StandardTokenizerFactory` añadiendo soporte para URLs y direcciones de correo electrónico, que se reconocerán como un único token.

- `solr.ICUTokenizerFactory`: Utiliza la biblioteca de internacionalización ICU. Este tipo de campo es necesario en idiomas que no utilizan espacios para separar palabras, como tailandés y hebreo.
- `solr.PatternTokenizerFactory`: Utiliza una expresion regular personalizada para separar *tokens*.

4.4. Filtros

Los filtros de Solr están asociados con las etapas de un proceso de extracción de información textual. A continuación se describen los más habituales.

- Normalización a minúsculas.

  La clase `solr.LowerCaseFilterFactory` convierte los campos que procesa a minúsculas para hacer búsquedas no sensibles a capitalización.

- Eliminación de *stopwords*.

  La clase `solr.StopFilterFactory` permite especificar un listado de palabras que serán ignoradas durante el análisis.

  La configuración de ejemplo de Solr incluye un listado de *stopwords* para el inglés. En Internet se pueden encontrar listas adaptadas para otros idiomas, incluido el castellano.

- Sustitución de sinónimos.

  Solr permite especificar un fichero de diccionario de sinónimos utilizando la clase `solr.SynonymFilterFactory`. El fichero de sinónimos es un fichero de texto cuya sintaxis es un conjunto de líneas, cada una definiendo un conjunto de palabras con un mismo significado, separadas por comas.

- Lematización.

  Solr implementa el algoritmo de bola de nieve para un amplio número de lenguajes. Para ello se utiliza la clase `solr.SnowballPorterFilterFactory` y se establece el atributo `language` con el nombre del idioma a utilizar, escrito en inglés.

- Normalización de caracteres especiales.

  La clase `solr.ASCIIFoldingFilterFactory` elimina los caracteres fuera del ASCII, como tildes o la letra eñe, y los sustituye por caracteres similares. La `ñ` se convierte en `n` y las tildes desaparecen.

  Esto es útil para búsquedas en español, donde muchos usuarios no introducen este tipo de caracteres. Alternativamente, Solr incluye otras clases para encontrar similitudes entre palabras en otros idiomas.

4.5. Definición de campos

El siguiente ejemplo define un esquema para los artículos de Wikipedia en español, recogiendo título, URL y un resumen opcional.

```
<field name="id" type="string" indexed="true" stored="true"
       required="true" multiValued="false" />
<field name="title" type="text_es" indexed="true" stored="true"
       required="true" multiValued="false" />
<field name="abstract" type="text_es" indexed="true" stored="true"
       required="false" multiValued="false" />
<field name="url" type="string" indexed="false" stored="true"
       required="true" multiValued="false" />
```

4.6. Campos copia

En algunos casos se quiere indexar el mismo dato de maneras distintas, pudiendo utilizar cada versión de forma diferente en las búsquedas. Por ejemplo, es posible que queramos una versión de un campo de texto donde no se haya utilizado el algoritmo de bola de lematización para dar una relevancia mayor a los resultados con concordancia exacta.

Para ello es posible utilizar la etiqueta XML `<copyField/>`, especificando un campo de origen y un campo de destino. El campo de destino deberá definirse a parte, y habitualmente tendrá un tipo distinto al campo de origen. Por ejemplo:

```
<field name="title" type="text_es" indexed="true" stored="true"
       required="true" multiValued="false" />
<field name="title_fw" type="text_es_full_words" indexed="true"
       stored="true" required="true" multiValued="false" />
<copyField source="title" dest="title_fw"/>
```

5. Añadir documentos

Los documentos se añaden a Solr a través de una interfaz HTTP. El punto de acceso por defecto es el siguiente:

`http://<servidor>/solr/<core>/update`

En la URL anterior, `<servidor>` es la dirección IP más puerto donde se aloja el servidor Solr. En el caso del servidor local por defecto, ésta es `localhost:8983`. `<core>` es el nombre del core de Solr que se configuró en los pasos iniciales.

La petición debe utilizar el método `POST`. En los argumentos de la URL se pueden especificar algunas opciones:

- `wt`: Establece el formato a utilizar en la petición. Puede ser `json` o `xml`.
- `commitWithin`: Establece cuánto tiempo en milisegundos se le dará a Solr para que guarde los cambios en disco.

El cuerpo de la petición será un mensaje JSON con el contenido `{"add": {"doc": <documento>}}`, donde `<documento>` es un objeto conteniendo pares clave-valor que se corresponden con los campos definidos en `schema.xml`.

El siguiente código Python enviaría a Solr un documento recibido por parámetro:

```
import json
import requests
session = requests.session()

def send_doc(doc):
    res = session.post(solr_url + '/update', params={
        'wt': 'json',
        'commitWithin': 10000,
    }, headers={
        'Content-Type': 'application/json',
    }, data=json.dumps({'add': {'doc': doc}}))
    if not res.ok:
        print('Solr Error!')
        try:
            print(res.json()['error']['msg'])
        except:
            print(res.text)
```

6. Consultar documentos

Para la búsqueda Solr ofrece otra interfaz HTTP, cuyo punto de acceso por defecto está en:

```
http://<servidor>/solr/<core>/select
```

El panel de administración de Solr incluye una página para probar esta interfaz, mostrada en la figura siguiente.

Las etiquetas en la página se corresponden con los nombres de claves que se utilizan en la interfaz HTTP. Estos son algunos de los más significativos.

- `q`: La cadena de búsqueda.
- `rows`: Número de resultados por página.
- `start`: Índice del primer resultado a devolver, utilizado para paginación.
- `fl`: Una lista de campos a devolver en los resultados de la búsqueda, separados por comas.
    - El campo virtual `score` contiene la relevancia de los resultados de búsqueda dada una consulta como un valor decimal numérico. Por defecto no se devuelve, salvo que se especifique en esta opción.

Como resultado de la petición devuelve un objeto JSON con un array `docs` que contiene los resultados de la búsqueda, representados por objetos con pares clave-valor correspondientes a sus campos.

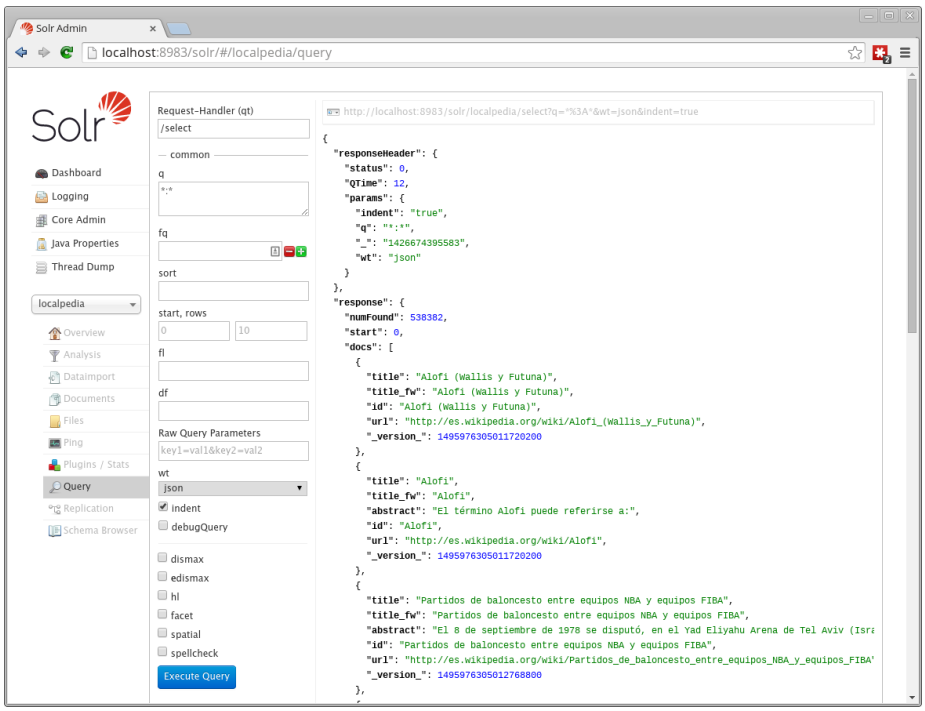

Figura 2: Búsqueda desde el panel de administración de Solr

6.1. Búsqueda natural

Por defecto, la consulta se interpreta de forma muy estricta, siguiendo la sintaxis de Apache Lucene.

Cuando las cadenas de búsqueda son escritas por humanos, se recomienda usar un parser más natural: `edismax`. Este parser no da errores de sintaxis, no necesita especificar los nombres de los campos en la cadena de búsqueda y soporta funcionalidades adicionales para la valoración de los resultados.

Para usar edismax se debe añadir a las opciones de búsqueda la propiedad `"defType": "edismax"`. Al hacerlo también se puede hacer uso de nuevas opciones de búsqueda:

- `qf`: Define los campos sobre los que se aplicará la búsqueda, separados por espacios. Si un campo no está definido aquí, no se considerará.

  Cada campo puede incorporar un *boost* para declarar que las coincidencias en ese campo son más importantes que en otros. Para ello se añade un sufijo `^x` donde `x` es un peso decimal; por ejemplo, `title^1.3`.

  Un campo para el que no se haya especificado un peso obtiene un valor de `1`.

  Pequeñas variaciones de los pesos pueden producir resultados de búsqueda muy distintos, por lo que deben ajustarse con cuidado.

- `pf`: Si todos los términos de búsqueda aparecen en alguno de los campos definidos en esta opción, el resultado afectado tendrá una bonificación positiva en su puntuación de relevancia.

  Generalmente usaremos aquí los campos de título, ya que una coincidencia total en el título es mucho más relevante que una mención al mismo dentro del texto de un documento.

Edismax soporta muchas más opciones, las cuales se pueden consultar en la documentación.

- `dismax` es un parser de búsqueda que es la base de `edismax`. En su página de documentación están detalladas la mayoría de opciones.
- En la página de documentación de `edismax` sólo se mencionan las opciones nuevas que aporta este módulo frente a `dismax`.

7. Puntos de acceso

En `solrconfig.xml` es posible definir puntos de acceso con parámetros de búsqueda por defecto.

```
<requestHandler name="/query" class="solr.SearchHandler">
   <lst name="defaults">
     <str name="echoParams">explicit</str>
     <str name="wt">json</str>
     <str name="indent">true</str>
     <str name="defType">edismax</str>
     <str name="qf">title^1.6 title_fw^1.5 abstract</str>
     <str name="pf">title_fw</str>
   </lst>
</requestHandler>
```

En este caso, `/query` haría una búsqueda utilizando por defecto el parser `edismax` con los pesos asignados en el ejemplo.

8. Demostración

La siguiente captura muestra la aplicación de ejemplo construida, haciendo una búsqueda sobre la palabra `calculo` en Wikipedia.

Nótese que se producen resultados satisfactorios pese a no haber usado tildes, gracias a la normalización de caracteres especiales. También, “cálculo” obtiene una relevancia superior a “calculadora” pese a que ambas palabras contienen la misma raíz.

Para esto se ha indexado el título en dos campos, haciendo uso de campos copia. En uno de los campos se ha utilizado el algoritmo de bola de nieve (campo `title`) y en otro no (campo `title_fw`). A la hora de hacer la búsqueda, ambos campos se tienen en cuenta al estar incluidos en el `qf` de `edismax`.

Figura 3: Demostración

9. Referencias

- Guía rápida de Solr:

  http://lucene.apache.org/solr/quickstart.html

- Analizadores, tokenizers y filtros:

  https://wiki.apache.org/solr/AnalyzersTokenizersTokenFilters

- Parámetros de búsqueda:

  http://wiki.apache.org/solr/CommonQueryParameters

- Soporte de idiomas:

  https://wiki.apache.org/solr/LanguageAnalysis

  http://svn.apache.org/repos/asf/lucene/dev/branches/lucene_solr_3_6/solr/example/solr/conf/schema.xml

- DisMax:

  https://cwiki.apache.org/confluence/display/solr/The+DisMax+Query+Parser

- EDisMax:

  https://cwiki.apache.org/confluence/display/solr/The+Extended+DisMax+Query+Parser

- Stopwords en varios idiomas, incluido el castellano:

  https://sites.google.com/site/kevinbouge/stopwords-lists