Preprocesado de documentos

Parte II. Análisis del texto

October 19, 2017

1 Objetivo

En esta práctica profundizaremos en el conocimiento del proceso de indexación, en concreto nos centraremos en cómo podemos utilizar Lucene para analizar un texto.

La práctica se realizará en grupo. Al final de la práctica se debe entregar un informe que necesariamente debe incluir una sección denominada *Trabajo en Grupo* en el que se indicará de forma clara la contribución de cada alumno. Se recomienda seguir la metodología SCRUM para el desarrollo del proyecto.

2 Apache Lucene

Apache-Lucene `https://lucene.apache.org/core/` es una biblioteca, escrita completamente en Java, y diseñada para la implementación de un motor de búsqueda de alto rendimiento. Actualmente se encuentra disponible para su descarga la versión 7.0.0. En el caso en que los documentos que queremos buscar dispongan de una estructura (por ejemplo, un email tiene subject, body, from, to, etc.) podemos aprovecharla para para mejorar la calidad de la búsquedas.

Lucene es el núcleo de servidores de búsqueda como Solr y Elasticsearch. También puede incorporarse a aplicaciones Java, Android o backends web.

Lucene ofrece potentes funciones a través de una sencilla API que permite:

- Indexación escalable y de alto rendimiento

  - Más de 150 GB / hora en hardware moderno
  - Requisitos de RAM pequeños - sólo 1 MB de Heap
  - Indexación incremental tan rápido como la indexación por lotes
  - El tamaño del índice es aproximadamente 20-30% del tamaño del texto indexado

- Algoritmos de búsqueda potentes, precisos y eficientes

  - Búsqueda ordenada - los mejores resultados devueltos primero
  - Consulta potentes: consultas de frases, consultas comodín, consultas de proximidad, consultas de rango y más
  - Búsqueda por campos (por ejemplo, título, autor, contenido)
  - Ordenación por cualquier campo
  - Búsqueda en múltiples índices
  - Permite la actualización y búsqueda simultáneas
  - Búsqueda por categorías (facetas), resaltado (highlighting), agrupación de resultados.
  - Incluye distintos modelos de búsqueda.

3 Procesado de textos con Lucene

La entrada de Lucene es un texto sin formato, y el primer paso del proceso de indexación consiste en realizar un análisis del contenido textual del documento para romper el texto en pequeños elementos de indexación - tokens. La forma en que el texto de entrada se divide en tokens influye en cómo la gente podrá buscar ese texto, por lo que constituye una parte esencial de todo proceso de indexación. Como es normal, el análisis es una de las principales causas de la indexación lenta. En pocas palabras, cuanto más se analiza, más lenta es la indexación (en la mayoría de los casos).

Lucene nos proporciona un conjunto de métodos que nos facilitan esta tarea. Este es el objetivo de esta práctica, donde utilizaremos la terminología Lucene.

Hay cuatro clases principales encargadas de realizar el análisis, aunque la relación entre ellas puede parecer confusa, por lo que es conveniente entender los siguientes elementos que forman parte del proceso del análisis.

- CharFilter: Extiende la clase Reader para transformar el texto antes de ser tokenizado, pero controlando los offsets (desplazamientos) de los caracteres corregidos. Pueden modificar el texto de entrada añadiendo, eliminando o transformando caracteres.

- Tokenizer: Es un TokenStream y es el responsable de romper el texto en tokens. Un token representa un término o palabra del documento, teniendo asociado elementos como su posición, offset de inicio y fin, tipo de token, etc.

  En algunos casos, basta con romper el texto de entrada considerando la separación por espacios en blanco, tabuladores o saltos de línea, pero en otros muchos es necesario un análisis más profundo. Por ejemplo, cuando deseamos construir N-gramas o se utilizan expresiones regulares para construir distintos tokens

- TokenFilter: Es un TokenStream y el responsable de modificar los tokens obtenidos por el Tokenizer pudiendo

  – Stemming - Sustitución de palabras con por sus raíces. Por ejemplo, en castellano, perro y perra se unen en un único término con raíz 'perr'.

  – Eliminación de palabras vacías - las palabras comunes como "el", "y" y "a" raramente agregan cualquier valor a una búsqueda. Su eliminación reduce el tamaño del índice y aumenta el rendimiento. También puede reducir el "ruido" y mejorar la calidad de la búsqueda.

  – Normalización de texto - Eliminar acentos y otras marcas de caracteres pueden mejorar la búsqueda.

  – Expansión de Sinónimos - Añadir sinónimos en la misma posición simbólica a la palabra actual puede ayudar a una mejor coincidencia cuando los usuarios buscan con palabras en el conjunto de sinónimas.

- Analyzer: Es el encargado de construir un TokenStream que pueda ser digerido por los procesos de indexación como de búsqueda. No se encarga de procesar el texto, para ello utiliza a elementos del tipo CharFilter, Tokenizer y TokensFilter.

  Si queremos utilizar una combinación concreta de CharFilter, Tokenizer y TokenFilter lo mas cómodo es crear una subclase de Analyzer. Sin embargo, antes de nada será conveniente estudiar aquellas que nos proporciona Apache Lucene ya implementadas, de entre las que podemos destacar el StandardAnalyzer, que es utilizado en muchas aplicaciones.

3.1 Ejemplos de Analyzer implementados en Lucene

Podemos encontrar distintos Analyzer ya implementados, donde cada uno tiene ejecuta una cadena distinta de pasos (CharFilter+Tokenizer+TokenFilter). La documentación de los distintos Analyzers la podemos encontrar en `https://lucene.apache.org/core/7_1_0/core/org/apache/lucene/analysis/Analyzer.html`. Entre ellos, destacaremos:

- KeywordAnalyzer: Considera el texto como un único token
- WhiteSpaceAnalyzer: Divide el texto considerando como separadores de tokens los espacios en blanco
- SimpleAnalyzer: Divide el texto separando por todo aquello que no sean letras y convierte a minúsculas.
- StopAnalyzer: Hace lo mismo que el SimpleAnalyzer pero elimina las palabras vacías (sin significado). Viene con una lista de palabras vacías predefinidas, pero como es obvio podemos darle las nuestras.
- StandardAnalyzer. Es el más elaborado, y es capaz de gestionar acrónimos, direcciones de correo, etc. Convierte a minúscula y elimina palabras vacías.
- UAX29URLEmailAnalyzer. Diseñado específicamente para trabajar con URL y direcciones de email, incluyendo además y filtrado de palabras vacías y conversión a minúsculas.

- Basados en Idiomas, permitiendo la eliminación de palabras vacías:
  - EnglishAnalyzer, en inglés.
  - SpanishAnalyzer, en castellano.
  - FrenchAnalyzer, en francés,
  - ...

También permite tratar los distintos campos de un texto de forma distinta utilizando un PerFieldAnalyzerWrapper. Así, por ejemplo, si queremos indexar emails, podríamos utilizar un StandadAnalyzer para el cuerpo del texto y un KeywordAnalyzer para los campos From y To, por ejemplo.

4 Estudio de Analyzers

Normalmente, cuando tratamos de indexar un documento o realizamos una consulta, nos limitamos a indicarle a Lucene qué analizador queremos utilizar de entre los que ya vienen construidos, o por el contrario decidimos crear uno propio. En cualquier caso, podemos considerar que en el uso normal de un analizador se toma como entrada el documento y su salida pasa directamente al proceso de indexación. Nosotros no llegamos a ser conscientes del resultado del análisis.

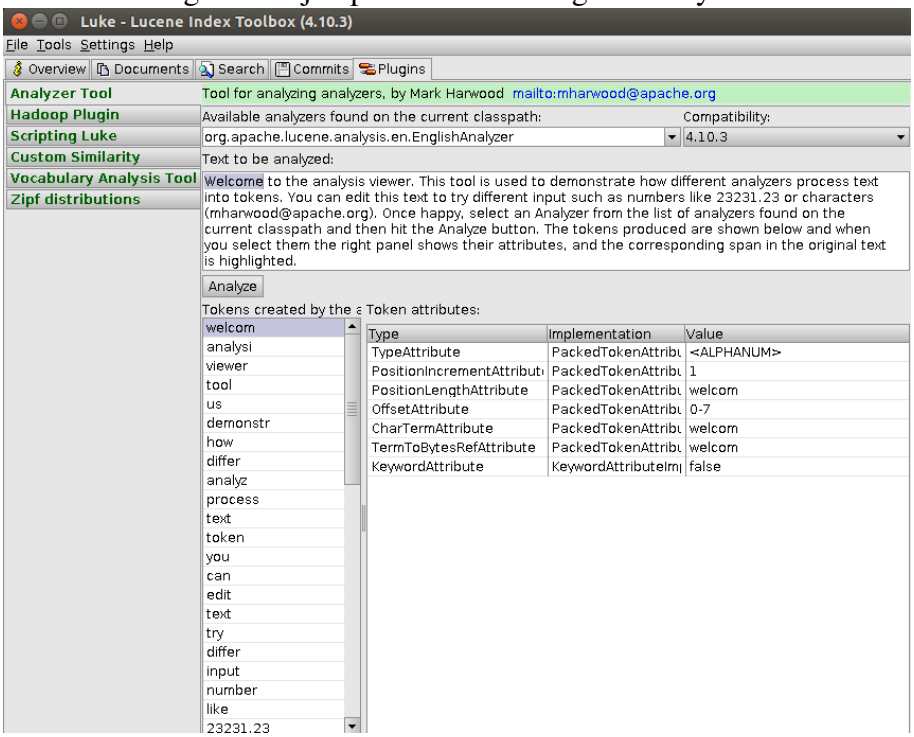
Si embargo, en esta práctica si queremos ver con más detalla el resultado del mismo. Será de utilidad para saber exactamente qué tokens serán indexados finalmente.

Con esta finalidad, y a modo de ejemplo, podemos utilizar el plugin Analyzer Tool de Luke: Luke nos proporciona una interfaz gráfica para ver un índice Lucene, pero en este caso nos centraremos en considerar el pluging Analyzer para ver que tokens son obtenidos por cada analizador.

Para ello, debemos descargarnos Luke (lo podemos encontrar en la página web de la asignatura) y ejecutar el fichero .jar.

Al ejecutar Luke, la primera opción es abrir un índice ya creado, pero nosotros no lo haremos en esta práctica, sino que nos vamos directamente a la pestaña Plugins y seleccionamos el Analyzer Tool. Una vez que estamos en esta ventana, ya podemos seleccionar el analizador que nos interesa dentro del desplegable y

Figure 1: Ejemplo de uso del EnglishAnalyzer con Luke.



proporcionar el texto que queremos analizar. Podremos ir viendo cada uno de los tokens generados, como nos muestra la imagen de la Figura 1.

4.1 Visitando un Analyzer

Una vez que hemos experimentado un poco con Luke, pasaremos a ilustrar como llamar a un Analyzer desde nuestros programas. Recordad que este proceso no es el normal en una etapa de indexación, sólo lo haremos para comprender mejor todo el proceso.

En la línea 4 se declara stream como una variable del tipo TokenStream, capaz de enumerar la secuencia de tokens que devuelve el analizador. El método addAttribute de TokenStream chequea si la instancia de la clase se encuentra en el TokenStream y la devuelve. En caso contrario, se crea una nueva instancia y la añade al token stream. Dicha instancia será posteriormente utilizada por los consultores para conocer información sobre los distintos tokens.

Las líneas 8 a 12 nos muestran la secuencia de pasos que tenemos que realizar

para explorar los resultados del analizador:

- stream.reset() Inicializa el stream, es necesario hacerlo antes de llamar a incrementToken. Se almacenan referencias locales a todos los atributos que se pueden acceder.
- stream.incrementToken(): Se posiciona en un atributo en cada llamada, devuelve falso cuando no hay mas atributos en el stream
- stream.end() Es necesario llamar a este método cuando se alcanza el final (incrementToken devuelve falso).

```
1  public static void tokenizeString(Analyzer analyzer, String
       string) {
2
3      try {
4        TokenStream stream  = analyzer.tokenStream(null, new
           StringReader(string));
5        OffsetAttribute offsetAtt = stream.addAttribute(
           OffsetAttribute.class);
6        CharTermAttribute cAtt= stream.addAttribute(
           CharTermAttribute.class);
7
8        stream.reset();
9        while (stream.incrementToken()) {
10         System.out.println( cAtt.toString()+" : ["+ offsetAtt.
             startOffset()+"," + offsetAtt.endOffset()+"]");
11       }
12       stream.end();
13     } catch (IOException e) {
14         throw new RuntimeException(e);
15       }
16 }
```

Para ver el efecto un un analizador sobre el texto tendremos que utilizar el siguiente código:

```
1    Analyzer an = new StandardAnalyzer( )
2    String text="This is a new text";
3    tokenizeString(an,text);
```

5 Creando nuestro propio analizador

Aunque Lucene dispone de múltiples Analyzers, a veces es necesario crear un analizador específico para una determinado tipo de texto de entrada. En general, este analizador puede requerir la llamada a distintos Tokenizer y TokenFilter.

Para definir el comportamiento del analizador, la subclases deben definir sus TokenStreamComponents como salida del método createComponents(String). Las distintas componentes serán utilizadas en las distintas llamadas del tokenStream(String, Reader).

Veamos el siguiente ejemplo simple que viene con la documentación de Lucene:

```
1  Analyzer analyzer = new Analyzer() {
2    @Override
3     protected TokenStreamComponents createComponents(String
         fieldName) {
4       Tokenizer source = new FooTokenizer(reader);
5       TokenStream filter = new FooFilter(source);
6       filter = new BarFilter(filter);
7       return new TokenStreamComponents(source, filter);
8     }
9     @Override
10    protected TokenStream normalize(TokenStream in) {
11      // Assuming FooFilter is about normalization and BarFilter
            is about
12      // stemming, only FooFilter should be applied
13      return new FooFilter(in);
14    }
15  };
```

6 Se pide

1. Sobre los documentos (libros del proyecto Gutenberg) utilizados en la práctica anterior, hacer un estudio estadístico sobre los distintos tokens que se obtienen al realizar distintos tipos de análisis. Por tanto, será necesario contar el número de términos de indexación así como frecuencias de los mismos en cada documento. Realizar un análisis comparativo entre los distintos re-

sultados obtenidos.

2. Implementar un analizador específico para indexar código fuente, donde serán delimitadores los espacios en blanco y símbolos que no sean letras o números, elimine todas las palabras reservadas del lenguaje y normalizando las etiquetas a minúscula. Para ello, la forma mas inmediata puede ser utilizar el CustomAnalyzer (`https://lucene.apache.org/core/7_1_0/analyzers-common/org/apache/lucene/analysis/custom/CustomAnalyzer.html`). Otra alternativa, es implementar nuestro propia clase Analyzer, como se indica en la documentación de Lucene, `https://lucene.apache.org/core/7_1_0/core/org/apache/lucene/analysis/package-summary.html`

3. **Sólo para los grupos de dos o mas miembros** Diseñar un analizador propio. Se os da libertad para poder escoger el dominio de ejemplo que consideréis mas adecuado, justificar el comportamiento.

6.1 Fecha de entrega

- 3 de Noviembre.