

Lucene.

Sistema de RI de
código abierto:
Búsquedas

Conceptos fundamentales

- Los conceptos fundamentales en Lucene son index, document, field y term.
- Un índice (index) contiene una secuencia de documentos.
 - ♦ Un Document es una secuencia de Fields.
 - ♦ Un Field es una secuencia Term.
 - ♦ Un Term es una secuencia de bytes.
- La misma secuencia de bytes en dos campos diferentes se considera un término diferente. Así términos se representan como un par: la cadena con el nombre del campo, y los bytes dentro del campo.

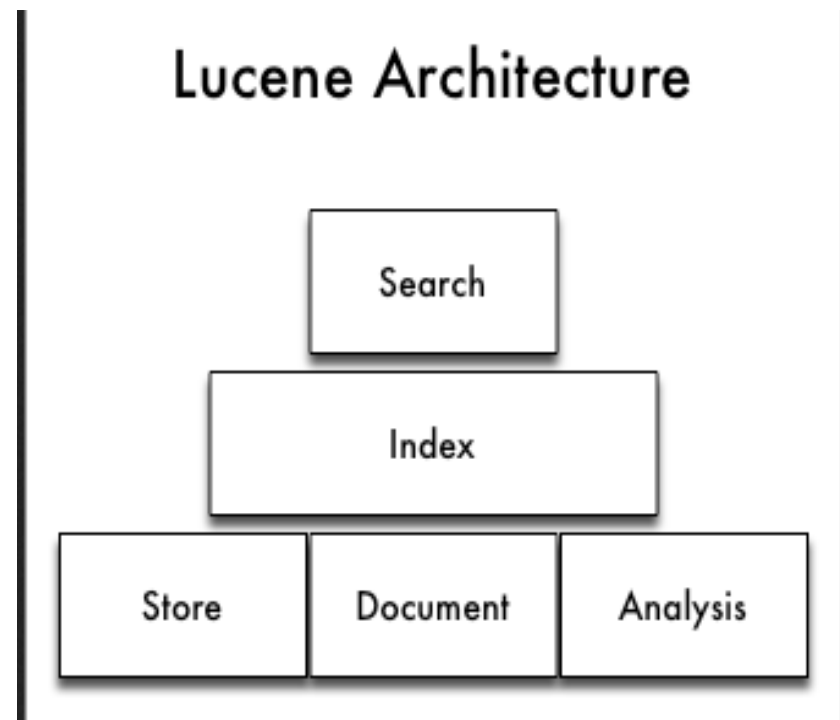
El índice Invertido

- El índice almacena estadísticas sobre los términos con el fin de hacer la búsqueda más eficiente. Dado un término se puede enumerar los documentos que lo contienen.
 - ♦ Tipos de campos:STORED, INDEXED, TOKENIZED
- Los índices de Lucene pueden estar compuestas de múltiples sub-índices, o **segmentos** . Cada segmento es un índice completamente independiente, puede ser buscado por separado.
- Los índices evolucionan a través de:
 - ♦ La creación de nuevos segmentos con nuevos docs
 - ♦ La fusión de segmentos existentes .
- Las búsquedas pueden incluir múltiples segmentos y/o varios índices, cada índice potencialmente compuesto por un conjunto de segmentos.

Recordemos

Para usar Lucene, una aplicación debería:

- Indexar:
 - Crear Documents añadiéndole Fields;
 - Crea un IndexWriter y añadir documentos con addDocument();
- Buscar:
 - Abrir el Índice
 - Llamar a QueryParser.parse() para construir una consulta desde un string
 - Buscar en el índice
IndexSearcher.search()



Apache Lucene: Indexación

```
Analyzer analyzer = new  
    StandardAnalyzer(Version.LUCENE_43);  
  
// Store the index in memory:  
Directory directory = new RAMDirectory();  
// en disco ... //Directory directory = FSDirectory.open("/tmp/testindex");  
IndexWriterConfig config = new  
    IndexWriterConfig(Version.LUCENE_43, analyzer);  
IndexWriter iwriter = new IndexWriter(directory, config);  
Document doc = new Document();  
String text = "Texto que podemos Indexar ( y luego BUSCAR)!! .";  
doc.add(new Field("name", text, TextField.TYPE_STORED));  
iwriter.addDocument(doc);  
iwriter.close();
```

Apache Lucene: Búsqueda

.....

```
DirectoryReader ireader = DirectoryReader.open(directory);
IndexSearcher isearcher = new IndexSearcher(ireader);
// Parse a simple query that searches for "text":
QueryParser parser = new
    QueryParser(Version.LUCENE_43, "name", analyzer);
Query query = parser.parse("Este es el texto a buscar en la coleccion");
ScoreDoc[] hits = isearcher.search(query, null, 1000).scoreDocs;
// Iterate through the results:
for (int i = 0; i < hits.length; i++) {
    Document hitDoc = isearcher.doc(hits[i].doc);
    System.out.println("salida "+hitDoc.get("name").toString());
    System.out.println("salida "+hitDoc.toString());
}
ireader.close();
directory.close();
```


Profundizando en la búsqueda

Photoshop PSD file download - Resolution 1280x1024 px - www.psdgraphics.com



Esquema de búsqueda

- La búsqueda requiere un índice ya construido.
- Crear una consulta, [Query](#) Lucene implementa una gran variedad de alternativas para la consulta, que se pueden combinar para construir consultas complejas usando información posicional.
 - Por lo general, a través de [QueryParser](#), [MultiPhraseQuery](#), etc que analiza la entrada del usuario
 - Es importante que las consultas usen el mismo [Analyzer](#) que el que se utilizó cuando se creó el índice.
- Leer el Índice, [IndexReader](#)
- Buscar en el Índice, por ejemplo a traver de [IndexSearcher](#)
 - [IndexSearcher.search \(Query, int\)](#)
 - [IndexSearcher.search \(Query, Filtro, int\)](#)
- Iterar a través de documentos devueltos, [Hits](#)
 - Extraer los elementos a mostrar al usuario, junto a los scores

Clases importantes en Búsqueda

- **IndexReader**
 - Lleva el índice a memoria
- **Searcher**
 - Proporciona los métodos de búsqueda
 - **IndexSearcher**, MultiSearcher, ParallelMultiSearcher
- **TopDocs**
 - Almacena los resultados de la búsqueda
- **QueryParser**
 - Gramática en JavaCC para crear las consultas
- **Query**
 - Representación lógica de las necesidades de inf.
- **Term**
 - Representa la unidad básica de búsqueda

Esquema del tema

- **Query:** Clases que manipulan la consulta Query
- **QueryParser:** Analiza la consulta del un usuario
- **IndexSearch:** Estudiaremos el proceso de búsqueda
- **IndexReader:** Analizaremos el Indice
- **Similarity:** Analizaremos la función de similaridad de lucene

Lucene Query

Clases de Consulta

- TermQuery
- BooleanQuery
- PhraseQuery
- TermRangeQuery
- NumericRangeQuery
- PrefixQuery
-



TermQuery

- Es el tipo de consulta más fácil de entender y el más utilizado en las aplicaciones. Una TermQuery devuelve todos los documentos que contienen el término (Term(“campo”,”texto”)) en el Field especificado
- La construcción de un TermQuery es tan simple como:

```
IndexSearcher searcher;
```

```
....
```

```
TermQuery tq = new TermQuery(new Term("fieldName", "term"));  
... searcher.search(tq, 5);
```

BooleanQuery

- Permite combinar multiples TermQuery en una consulta booleana, mediante el uso de cláusulas BooleanClauses, donde cada cláusula contiene una sub-consulta (instancia Query) y un operador (BooleanClause.Occur) que describe cómo ese sub-consulta se combina con las otras cláusulas:
 - ♦ SHOULD – Es un O-lógico.
 - ♦ MUST - Y-lógico.
 - ♦ MUST_NOT Negación

```
BooleanQuery booleanQuery = new BooleanQuery();  
booleanQuery.add(query1, BooleanClause.Occur.MUST);  
booleanQuery.add(query2, BooleanClause.Occur.MUST);
```

PhraseQuery

- Una consulta en la que los documentos relevantes deben contener una secuencia particular de términos.
- Esta consulta puede ser combinado con otros términos o consultas con un BooleanQuery.
- Una PhraseQuery se construye normalmente a través de QueryParser, tomando como entrada una cadena entre comillas como "Nueva York".
- Podemos controlar el número de palabras que se pueden permitir entre los términos en la frase, utilizando setSlop:
 - ♦ 0 indica búsqueda exacta (por defecto)

```
PhraseQuery phraseQuery = new PhraseQuery();  
phraseQuery.add(term1); phraseQuery.add(term2); //"term1 term2"  
phraseQuery.setSlop(slop);  
.... = indexSearcher.search(phraseQuery);
```


TermRangeQuery

- Una consulta cuyos documentos relevantes contienen los términos dentro de un rango (utilizan como criterio Byte.compareTo (Byte))
- No vale para comparaciones numéricas!!!

Por ejemplo,

```
TermRangeQuery trq;
```

```
trq = trq.newStringRange("filename","cap100.txt","cap120.txt",true,false);
```

- `TermRangeQuery(String field, BytesRef lowerTerm, BytesRef upperTerm, boolean includeLower, boolean includeUpper)`
- `TermRangeQuery newStringRange(String field, String lowerTerm, String upperTerm, boolean includeLower, boolean includeUpper)`

`ByteRef` (Lucene usa esta clase para representar datos en UTF8)

NumericRangeQuery

- Una consulta que coincide con los valores numéricos dentro de un rango especificado.
- Para utilizarla, es necesario que los campos hayan sido indexados considerando valores numéricos (utilizando IntField, floatfield, longfield o DoubleField).

- Ejemplo:

```
Q = Query NumericRangeQuery.newFloatRange ("peso", 0.03f, 0.10f, true, true);
```

Devuelve los documentos con valor de “peso” (flotante” este dentro de los rangos 0,03-0,10, inclusive.

- El rendimiento de NumericRangeQuery es mucho mejor que el TermRangeQuery debido a que el número de elementos que deben ser buscados suele ser mucho menor

Importancia del campo **Boost**

- Lucene permite influir en los resultados de una búsqueda
 - ♦ En tiempo de indexación: llamando `Field.setBoost ()` antes de que un documento se añada al índice.
 - ♦ Impulso en tiempo de consulta: llamando a `Query.setBoost ()` para impulsar una cláusula
- Multiplica por el `getBoost()` todos los emparejamientos
- El boost también se tiene en cuenta a la hora de computar la norma del documento (para normalizar el score)

QueryParser

Parse::



QueryParser

- Aunque Lucene proporciona la capacidad de crear sus propias consultas, también proporciona un lenguaje de consulta a través de QueryParser (Analizador de consultas) , un analizador léxico transforma una cadena (string) en una consulta Lucene
- Está especialmente diseñado para consultas dadas por el usuario a través de una cadena, string. Si la consulta se genera “programándola” es mejor utilizar las API de consulta.
- Los campos NOTOKENIZADOS, como por ejemplo fechas, claves, campos con un conjunto limitado de valores (generados por un desplegable) , etc es mejor añadirlos directamente a la consulta. (No deben pasar por el analizador)
- La sintaxis analizador de consultas varía entre versiones de Lucene

QueryParser

- Métodos más importantes:
- Constructor:

`QueryParser(Version matchVersion, String f, Analyzer a)`

matchVersion – Versión de Lucene

f – el Field por defecto sobre el que se busca

a – Analizador utilizado para indexar

- `Query parse(String cadena)`

Devuelve la consulta Lucene representada en cadena. Cadena debe estar escrita en el lenguaje de consulta Lucene

Expresiones válidas

Table 3.2 Expression examples that `QueryParser` handles

Query expression	Matches documents that...
<code>java</code>	Contain the term <i>java</i> in the default field
<code>java junit</code> <code>java or junit</code>	Contain the term <i>java</i> or <i>junit</i> , or both, in the default field ^a
<code>+java +junit</code> <code>java AND junit</code>	Contain both <i>java</i> and <i>junit</i> in the default field
<code>title:ant</code>	Contain the term <i>ant</i> in the <code>title</code> field
<code>title:extreme</code> <code>-subject:sports</code> <code>title:extreme</code> <code>AND NOT subject:sports</code>	Have <i>extreme</i> in the <code>title</code> field and don't have <i>sports</i> in the <code>subject</code> field
<code>(agile OR extreme) AND methodology</code>	Contain <i>methodology</i> and must also contain <i>agile</i> and/or <i>extreme</i> , all in the default field
<code>title:"junit in action"</code>	Contain the exact phrase " <i>junit in action</i> " in the <code>title</code> field
<code>title:"junit action"~5</code>	Contain the terms <i>junit</i> and <i>action</i> within five positions of one another
<code>java*</code>	Contain terms that begin with <i>java</i> , like <i>javaspaces</i> , <i>javaserver</i> , and <i>java.net</i>
<code>java~</code>	Contain terms that are close to the word <i>java</i> , such as <i>lava</i>
<code>lastmodified:</code> <code>[1/1/04 TO 12/31/04]</code>	Have <code>lastmodified</code> field values between the dates January 1, 2004 and December 31, 2004

^a The default operator is *OR*. It can be set to *AND* (see section 3.5.2).



Búsquedas simples:

<code>free AND "text search"</code>	Search for documents containing "free" and the phrase "text search"
<code>+text search</code>	Search for documents containing "text" and preferentially containing "search"
<code>giants -football</code>	Search for "giants" but omit documents containing "football"
<code>author:gosling java</code>	Search for documents containing "gosling" in the author field and "java" in the body

Lucene



Búsquedas con caracteres comodines:

- ? = Un único carácter (te?t = {text, test}).
- * = Varios caracteres (test* = {test, tests, tester}).
- Se pueden emplear combinados y como primer carácter del término.

Lucene



Búsquedas vagas:

- Se emplea el símbolo "~" al final de un término. Indica “parecido a”, “similar a” (usa la distancia Damerau-Levenshtein (emparejamiento optimal de strings))
- Ejemplo: `roam~` devolvería “foam” y “roams”.
- Se puede indicar un número real entre 0 y 1. Cuanto más próximo sea, más similar será con el término (`roam~0.8`). Por defecto, 0.5.

Lucene



Búsquedas por proximidad:

- Puede encontrar documentos en los que los términos especificados estén una distancia concreta unos de otros:
- Por ejemplo, "Jakarta apache"~10

Lucene



Consultas por rango:

- Permite hacer la correspondencia de documentos cuyos valores de los campos especificados estén entre un límite inferior y otro superior.
- Por ejemplo: `mod_date:[20020101 TO 20030101]` ó `title:{Aida TO Carmen}`
- Los límites exclusivos se nota con (y), mientras que los inclusivos con [y].

Lucene



Aumento de la importancia de un término en la consulta:

- Para incrementar la relevancia de un término para el usuario se utiliza el operador “^” que sigue al término y un número. Cuanto más alto sea, más importancia tendrá el término en la búsqueda.
- Por ejemplo, en lugar de la consulta: Jakarta apache
- Podemos indicar que le damos más importancia a “jakarta” mediante: Jakarta^4 apache.
- Los documentos donde aparezca “jakarta” serán más relevantes.
- También se pueden aumentar expresiones.



Para responder a cada una de estas consultas podemos utilizar la clase

IndexSearch

que es la que se encarga de computar el score para cada doc.

Realizar las consultas en Lucene es extraordinariamente rápido y esconde la casi totalidad de la complejidad al usuario.

En pocas palabras, funciona.

Clases importantes en Búsqueda

- Search
 - Clase abstracta que implementa los métodos principales de búsqueda.

IndexSearcher

- Implementa las búsquedas sobre un IndexReader .
El constructor recibe un IndexReader

```
IndexSearcher searcher = new IndexSearcher(reader);
```
- Una aplicación típica sólo necesita llamar a los métodos
 - search(Query q ,int n)
 - search(Query q,Filter f, int n)
- Se recomienda abrir un único IndexSearcher y utilizarlo en todas las búsquedas

```
Import org.apache.lucene.search.IndexSearcher;
```

IndexSearch: Algunos métodos

Además de buscar, vía **search(...)**, la clase IndexSearch nos permite considerar componentes de una búsqueda

- **Document doc(int docID):** Nos devuelve del índice el documento docID
- **Explanation explain(Query q, int docId)** Nos explica cómo se computa el score de la consulta q para el docID
- **void setSimilarity(Similarity s):** Nos permite indicar la función de similaridad que se utiliza para calcular el score
 - TFIDFSimilarity (version del modelo de espacio vectorial)
 - BM25Similarity,
 - LMDirichletSimilarity (Language Model)
 - PerFieldSimilarityWrapper (usa una similaridad por campo)

Clases importantes en Búsqueda:

TopDocs

- Estructura que almacena los resultados de la búsqueda, representa una array ordenado de documentos, según relevancia a la consulta.
- Atributos de TopDocs:
 - ♦ **ScoreDocs** : Vector con los top hits
 - Un hits contiene un
 - Doc, numero de doc
 - Score el score de este doc
 - ♦ **TotalHits**: El numero total de hits

Ejemplo Search/TopDocs

```
TopDocs hits = searcher.search(query, 15);
```

```
for (int i=0; i< hits.totalHits ; i++){  
    int docId = hits.scoreDocs[i].doc;  
    Document d = searcher.doc(docId);  
    Float r = hits.scoreDocs[i].score  
    System.out.println( r + d.get("filename"));  
}
```

Imprime los 15 docs con mejor score

```
System.out.println("Explaining "+searcher.explain(query,  
docId).toString());
```


Personalizando las búsquedas....

- Podemos ordenar la salida considerando cualquier sobre los datos almacenados.

`TopFieldDocs search(Query query, int n, Sort sort)`

- Devuelve los top n docs, ordenados segun sort
 - ♦ Los campos utilizados para determinar el orden deben tener un único término, que determina la posición relativa en el orden
 - ♦ El campo debe indexarse, NO tokenizarse, y no necesita estar almacenado
- Atributos de TopFieldsDocs
 - ♦ `totalHits`
 - ♦ `ScoreDocs`
 - ♦ `SortField[] fields`; campos utilizados para ordenar

Ejemplo

```
Sort orden = new Sort();  
orden.setSort(new SortField("size",SortField.Type.LONG));
```

```
TopFieldDocs tfds = searcher.search(q, 5, orden);
```

```
ScoreDoc[] hits_tama = tfds.scoreDocs;  
System.out.println("Found " + hits_tama.length + " hits.");  
for(int i=0;i<hits_tama.length;++i) {  
    int docId = hits_tama[i].doc;  
    Document d = searcher.doc(docId);  
    System.out.println((i + 1) + ". " + d.get("path") + " score=" +  
        hits_tama[i].score+" sz " +d.get("size"));  
}
```

Facetas (categorías) en Lucene

- Una faceta (categoría) puede ser utilizado por lucene para clasificar documentos
 - ♦ Libros: autor, precio,
- En una búsqueda por facetas, además del conjunto estándar de resultados de búsqueda, también obtenemos listas por subcategorías.
 - ♦ Por ejemplo, para la faceta autor faceta, se obtiene una lista de autores relevantes,
 - ♦ Cuando los usuarios hacen clic en estas subcategorías, se restringe la búsqueda
- En esencia, la búsqueda por facetas hace facilita la navegación a través de los resultados de búsqueda.
- Útil en e-comercio.

Shop by
Department ▾

Search All ▾ samsung

Go

Hello
You

Departments

Cell Phones & Accessories

- Unlocked Cell Phones
- Cell Phone Cases
- Smart Watches & Accessories

Electronics

- Televisions
- LED TVs
- LCD TVs
- Audio & Video Accessories
- Televisions & Video Products

Computers & Accessories

- Computer Tablets
- Laptop Computers
- Computer Components

+ See All 35 Departments

Eligible for Free Shipping

Free Shipping by Amazon

Brand

- ☐ Samsung
- ☐ Generic
- ☐ eForCity
- ☐ ULAK
- ☐ Importer520
- ☐ Empire
- ☐ Fosmon Technology
- ☐ MoKo
- ☐ Skinit
- ☐ Fintie
- ☐ MyBat

Gifts from \$25



"samsung"

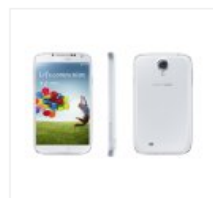
Related Searches: [samsung galaxy s4](#), [samsung s4](#), [samsung tv](#)

Showing 1 - 16 of 9,299,531 Results

Amazon's Samsung Store



Samsung

Unlocked Cell
PhonesCell Ph
Acces

Samsung Galaxy Tab 3 (7")

~~\$199.99~~ Click to see price Prime
In stock on December 20, 2013

More Buying Choices
\$149.99 used & new (57 offers)

Narrow results:

608 hotels

Name contains

Hotel name...

Price (total)

\$0 to \$500+



Star rating

- ☐ ★★★★★
- ☐ ★★★★
- ☐ ★★★
- ☐ ★★
- ☐ ★

Guest rating

0 to 5



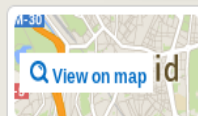
Neighbourhood

- ☐ Gran Via
- ☐ Puerta del Sol
- ☐ Barajas
- ☐ Salamanca

Hotels.com®
Wake Up Happy®

Home Deals Welcome Rewards®

Customize



Madrid, Spain Sat 14 - Sun 15, December 2013, 1 night 1 room, 2 adults,

Most popular

Star ratings

Distance

Guest rating

Hotel Atlantico Madrid

Last booked 13 hours ago

Gran Via 38 Madrid, Madrid, 28013 Spain, 0044 203 564 5228



★★★★★

Gran Via

0.23 mi to city centre

Welcome Rewards

Outstanding
4.6 / 52,046
customer reviews

Regina

Last booked 7 hours ago

Alcala 19 Madrid, Madrid, 28014 Spain, 0044 203 564 5228



★★★★★

Gran Via

0.21 mi to city centre

Welcome Rewards

Excellent
4.3 / 51,293
customer reviews

Apartamentos Recoletos

Last booked 17 hours ago

Calle de Villanueva, 2 Madrid, Madrid, 28001 Spain, 0044 203 564 5228



★★★★★

Salamanca

0.79 mi to city centre

Welcome Rewards

Good
3.3 / 532
customer reviews

Indexar

- Para buscar por facetas es necesario indexar previamente
- Para cada documento de entrada:
 - Crear un nuevo Documento Lucene
 - Analizar el texto de entrada y agregar los campos de búsqueda de texto apropiados
 - Obtener las categorías asociadas al documento y crear un CategoryDocumentBuilder con la lista de categorías
 - Construir el documento - esto agrega las categorías al documento Lucene.
 - Añadir el documento al índice

Ejemplo

```
IndexWriter writer = ...
TaxonomyWriter taxo = new DirectoryTaxonomyWriter(taxoDir,
OpenMode.CREATE);
...
Document doc = new Document();
doc.add(new Field("title", titleText, Store.YES, Index.ANALYZED));
...
List<CategoryPath> categories = new ArrayList<CategoryPath>();
categories.add(new CategoryPath("author", "Mark Twain"));
categories.add(new CategoryPath("year", "2010"));
...
DocumentBuilder categoryDocBuilder = new CategoryDocumentBuilder(taxo);
categoryDocBuilder.setCategoryPaths(categories);
categoryDocBuilder.build(doc);
writer.addDocument(doc);
```

Búsqueda ... (agrupando facetas)

- Usar Facetas permite ver los resultados de búsqueda como
 - Un conjunto de documentos - un subconjunto de los documentos del índice que coincidan con la consulta
 - Analizando las Facetas - Por ejemplo, contar con una cierta dimensión de la faceta.
- FacetRequest es un componente básico en la búsqueda por facetas
Cada solicitud de faceta tiene al menos dos campos:
 - CategoryPath - categoría raíz de la solicitud. Las categorías que se devuelven como resultado de la solicitud, toda serán descendientes de esta raíz
 - Número de Resultados - número de sub-categorías a devolver volver (como máximo).³⁹

Ejemplo

```
IndexReader indexReader = DirectoryReader.open(indexDir);
IndexSearcher searcher = new IndexSearcher(indexReader);
TaxonomyReader taxo = new DirectoryTaxonomyReader(taxoDir);
...
Query q = new TermQuery(new Term(SimpleUtils.TEXT, "white"));
TopScoreDocCollector tdc = TopScoreDocCollector.create(10, true);
...
FacetSearchParams facetSearchParams = new FacetSearchParams();
facetSearchParams.addFacetRequest(new CountFacetRequest(
    new CategoryPath("author"), 10));
...
FacetsCollector facetsCollector = new FacetsCollector(facetSearchParams,
indexReader, taxo);
    searcher.search(q, MultiCollector.wrap(topDocsCollector,
facetsCollector));
    List<FacetResult> res = facetsCollector.getFacetResults();
```

Profundicemos en el Indice...

IndexReader



- Como hemos visto, Lucene nos permite desarrollar un sistema de recuperación de forma eficiente

En pocas palabras, funciona.

Por lo menos, hasta que no funciona, o no funciona como es de esperar que funcione. Entonces nos queda más remedio que ver lo que está pasando por dentro

La clase IndexReader

- Una clase que proporciona los métodos básicos para acceder al índice y a los Fields almacenados cuando se muestra la lista de resultados.
- Desde la versión 4.0 NO es posible recuperar términos o lista de ocurrencias a través del índice. Para acceder a ellos se debe hacer a través de las subclases
 - ♦ AtomicReader (atómico)
 - ♦ CompositeReader (múltiple readers)
- Para crear una instancia de un IndexReader sobre un índice en disco se construyen haciendo la llamada a método DirectoryReader.open(), se le puede pasar al IndexSearcher.
- Los documentos en la clase IndexReader se identifican por un entero, (docId)

```
Import org.apache.lucene.index.IndexReader
```

Ejemplo de uso

```
String dir = "Colecciones/index_ObrasCervantes";
```

```
...
```

```
IndexReader reader = DirectoryReader.open(FSDirectory.open(new  
File(dir)));
```

```
IndexSearcher searcher = new IndexSearcher(reader);
```

```
....
```

```
Document d = reader.document(docId);
```

```
reader.numDeletedDocs()
```

```
Int n = reader.numDocs()
```

```
reader.close();
```

Estructura del Índice en Lucene

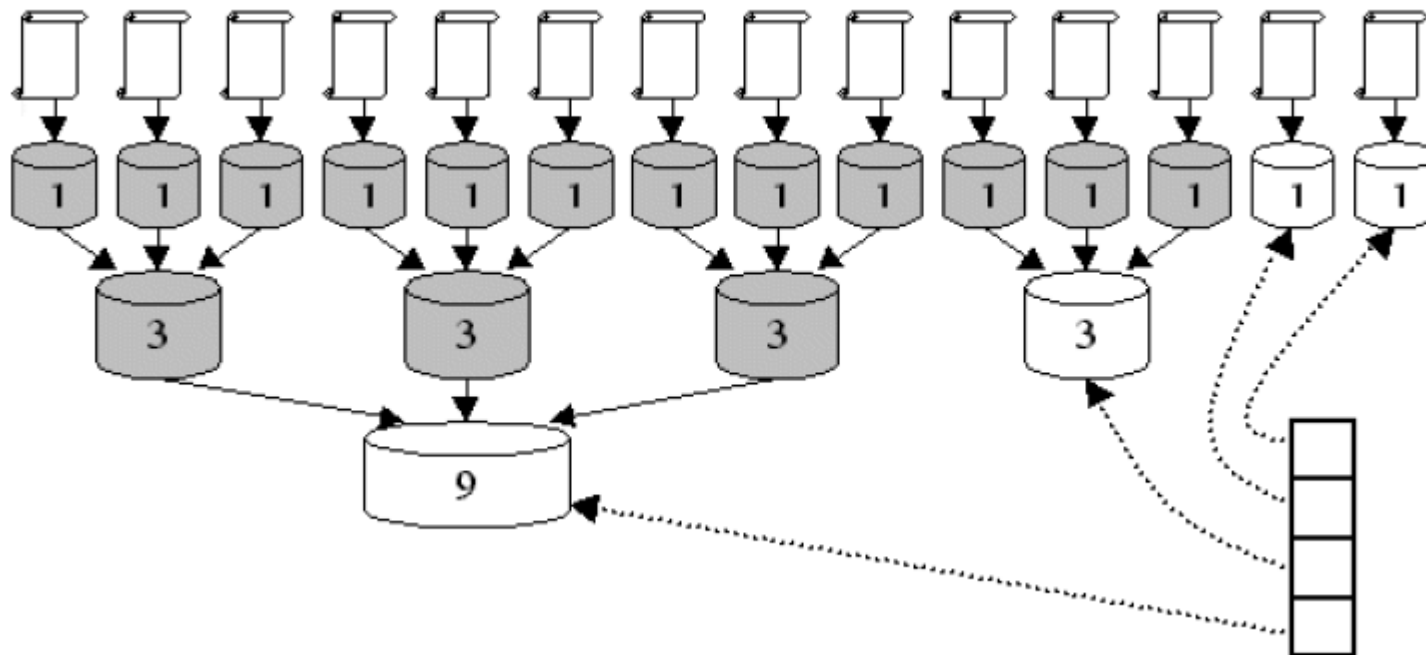
- Lucene fue el primer motor de búsqueda que soportó añadir y actualizar documentos, garantizando la coherencia del índice.
- Para ello, utiliza
 - ♦ una estructura de índices por segmentos
 - ♦ Al hacer un cambio en el índice se crean nuevos segmentos en Lucene.

Elementos de un Segmento:

- **Información de segmento.** metadatos como el número de documentos, archivos que utiliza, etc.
- **Nombres de campo**
- **Stored fields:** para cada documento , una lista de pares field – valor. Util para url, nombre fichero, etc. Clave DocId
- **Diccionario de términos.** Un diccionario que contiene todos los términos utilizados en todos los campos de todos los documentos. El diccionario también contiene el número de documentos que contienen el término, y los punteros a la frecuencia del término y los datos de proximidad.
- **Frecuencia de término.** Los documentos que contienen ese término, y la frecuencia del término en este documento
- **Datos de proximidad del término.** las posiciones que en las que aparece
- **Factores de normalización.** Para cada campo de cada documento, se guarda un valor de normalizacion (se multiplica por el score tras consulta)
- **Term Vector (opcional).** Para cada campo en cada documento, el término vector (a veces llamado vector documento)
- **Los documentos eliminados.**

Segmentos en Lucene

- Cada índice se compone de varios segmentos colocados en el directorio de índice.
- Todos los documentos son añadidos a nuevos segmentos, los cuales se mezclan (fusionan) con otros archivos en disco
- Se escriben los segmentos de forma incremental y se mezclan
- Optimizar un índice => un segmento









Visualización de mezcla de segmentos con Lucene.







- Indexando la Wikipedia en Inglés
- Video en: Mike McCandless' blog,
- <http://goo.gl/kI53f>

Los índices Lucene (hasta ver. 3.6)













- Cada segmento ("índice atómico") es un índice completamente funcional:
 - SegmentReader implementa la interfaz IndexReader para segmentos individuales
- Los índices compuestos:
 - DirectoryReader implementa la interfaz IndexReader sobre un conjunto de SegmentReaders
 - Multireader es una abstracción de múltiples IndexReaders combinado en un índice virtual
 - Term Dictionary: sobre la marcha mezcla y ordena los términos del índice
 - Postings: se añaden las listas de ocurrencias para cada término, haciendo los identificadores de doc globales

Mezcla de terminos y posting

Term 1	
Term 2	
Term 4	
Term 7	
Term 8	
Term 9	

Term 1	
Term 3	
Term 5	
Term 6	
Term 8	
Term 9	



Term 1		
Term 2		
Term 3		
Term 4		
Term 5		
Term 6		
Term 7		
Term 8		
Term 9		

Search antes de la ver. 2.9

- IndexSearcher utilizó los índices subyacentes siempre como un índice "único" (atómico):
 - ♦ Las consultas se ejecutan en la visión atómica de un índice compuesto
 - ♦ Ralentiza las consultas que recorren el diccionario de términos (MultiTermQuery) o involucran a muchos documentos,
=> Se recomienda "optimizar" el índice
 - ♦ En cada cambio, la FieldCache utilizada para ordenar se tiene que recargar completamente

Lucene 2.9 y siguientes

- Búsqueda se ejecuta de forma separada para cada segmento del índice, la vista “única” no es considerada en versiones posteriores!!
- Ventajas:
 - ♦ No realiza mezcla de los términos del diccionario
 - ♦ Posibilidad de paralelizar
 - ♦ Ordenar sólo necesita una FieldCache por segmento
 - Menos costosa de reabrir después de los cambios del índice!

Lucene 4.X: IndexReader

- Es la clase abstracta más general que proporciona la interfaz para acceder a un índice.
 - ♦ Superclase de tipos más específicos
 - ♦ No tiene constructor público
- NO sabe nada sobre el concepto de "índice invertido", NO tiene acceso a términos, ni posting, ... etc!
- Permite el acceso a los Fields almacenados por el docID: Útil para mostrar sólo los resultados de la búsqueda
- Permite conocer algunos metadatos muy limitados: Número de documentos, ...
- No permite modificaciones (borrados)
- Puede pasarse como argumento a IndexSearcher, como siempre
- Permite IndexReader.open () para la compatibilidad con versiones anteriores (en desuso)

Ejemplo ...

```
String index = “.....”;
```

```
IndexReader reader = IndexReader.open(FSDirectory.open(new  
File(index) )); // No es correcta actualmente (DEPRECATED)
```

```
Document d = reader.document(docID);
```

```
Int n = reader.numDocs();
```

```
If (reader.hasDeletions() ) {....}
```

```
IndexSearcher searcher = new IndexSearcher(reader);
```

```
Query q = ....
```

```
resultados = searcher.search(query, ...);
```

```
....
```

Tipos de IndexReaders

Hay dos tipos distintos de IndexReaders

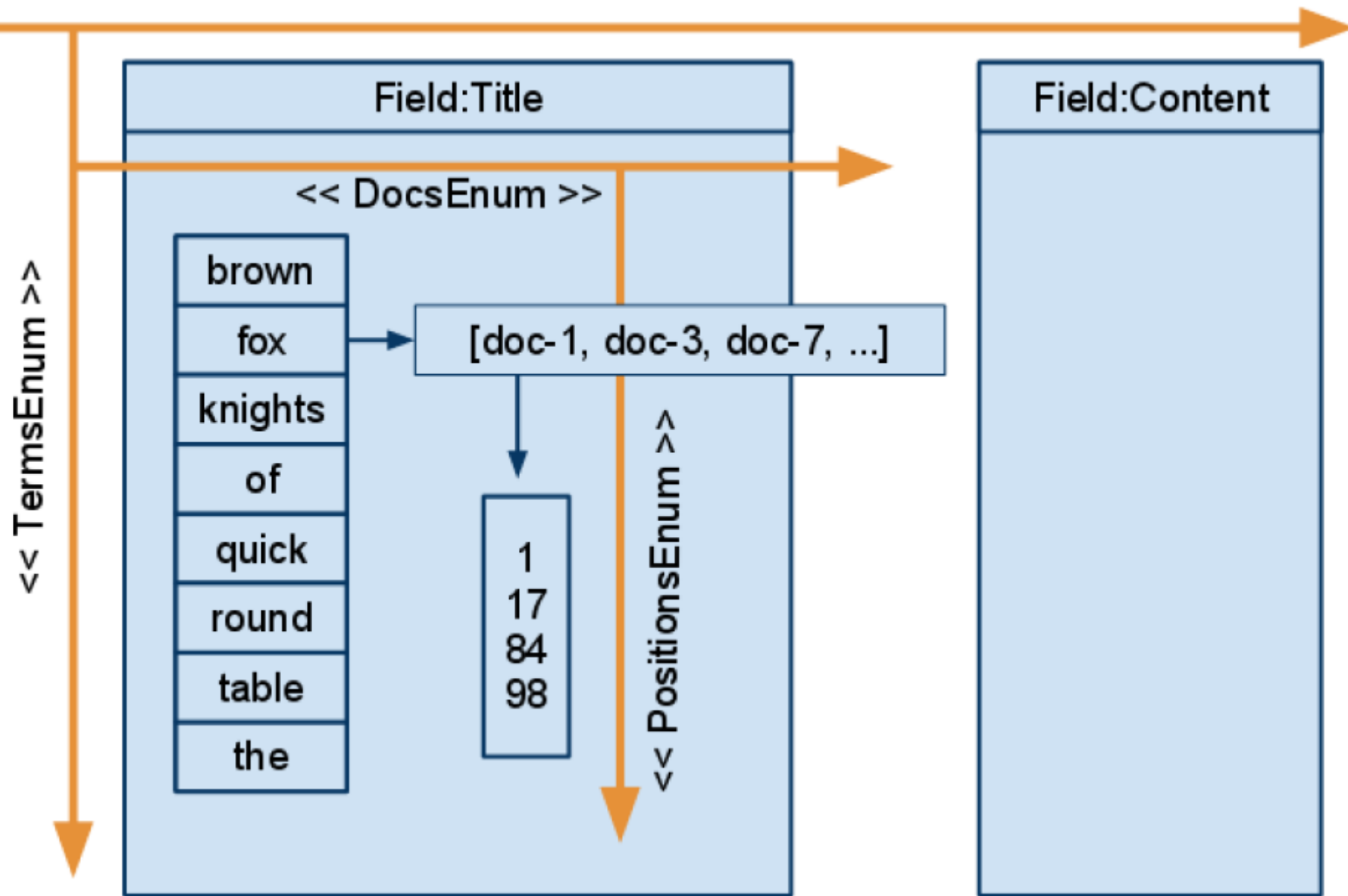
- **AtomicReader.** Estos índices son atómicos, permite el recuperar los Fields almacenados, los valores de doc, terminos y listans de ocurrencias.
- **CompositeReader:** Es un conjunto de AtomicReaders.
 - ♦ Una instancia de un CompositeReader (por ejemplo, DirectoryReader) sólo tiene acceso a los Fields almacenados en los distintos AtomicReaders.
 - ♦ Para construir un IndexReader de un índice en disco, usualmente se utiliza la llamada a **DirectoryReader.open()**

AtomicReader

- Hereda de IndexReader, permite el acceso al índice
- Controla el acceso a un índice atómico (segmentos individuales)
 - ♦ Permite recuperar los campos almacenados, docs, términos y listas de postings

AtomicReader

<< FieldsEnum >>



AtomicReader

- Mecanismos de acceso
 - ♦ **Fields** `fields()` Returns Fields for this reader.
 - ♦ **Terms** `terms(String field)` This may return null if the field does not exist.
 - ♦ **DocsEnum** `termDocsEnum(Term term)` Returns DocsEnum for the specified term.
 - ♦ **DocsAndPositionsEnum** `termPositionsEnum(Term term)` Returns DocsAndPositionsEnum for the specified term.
 - ♦ **Long** `getSumDocFreq(String field)`
 - ♦

CompositeReader

- No tiene funcionalidades adicionales sobre IndexReader
- Proporciona getSequentialSubReaders () para recuperar todos los Readers hijos
- Esta clase se implementa como
 - ♦ DirectoryReader
 - ♦ Multireader

DirectoryReader

- Clase abstracta, define la interfaz para:
 - Acceso a los índices en disco (sobre la clase Directory)
 - Acceso a los los metadatos de índice, la versión del índice, IsCurrent () para reabrir
 - Define openIfChanged (por reapertura de índices)
 - Los hijos son siempre AtomicReader

Ejemplo Básico de búsqueda ...

- Es similar al IndexReader

```
DirectoryReader reader = DirectoryReader.open(directory);
IndexSearcher searcher = new IndexSearcher(reader);
Query query = new QueryParser("fieldname", analyzer).parse("text");
TopDocs hits = searcher.search(query, 10);
ScoreDoc[] docs = hits.scoreDocs;
Document doc1 = searcher.doc(docs[0].doc);
// alternative:
Document doc2 = reader.document(docs[1].doc);
```

DirectoryReader

- Que debo hacer si necesito acceder a los términos y listas de posting de un DirectoryReader! ...
 - ♦ Optimizar el índice de forma que sólo tenga un segmento?

DirectoryReader

- Que debo hacer si necesito acceder a los terminos y listas de posting de un DirectoryReader! ...
 - ♦ Optimizar el índice de forma que sólo tenga un segmento? **NO OPTIMIZAR**
 - ♦ Optimizar implicaría la mezcla del índice
 - ♦ Mejor:
 - Acceder a los índices atómicos (hojas – leaves) reader.leaves()
 - Iterar sobre los sub-readers, posiblemente en paralelo,
 - Mezclar los resultados (no el índice)

DirectoryReader Ejemplo

```
String indexLocation = "....../index_Quijote";
DirectoryReader reader = DirectoryReader.open(FSDirectory.open(new File(indexLocation)));

System.out.println("SEGs"+reader.leaves().size());
for (AtomicReaderContext rc : reader.leaves()) {
    AtomicReader ar = rc.reader();

    FieldInfos fis = ar.getFieldInfos();
    for (FieldInfo fi : fis)
        System.out.println(fi.name+" " +fi.getIndexOptions()+fi.getNormType());

    Terms terminos = ar.terms("contents");
    TermsEnum te = terminos.iterator(TermsEnum.EMPTY);
    while( te.next() != null) {
        System.out.println("term=" + te.term().utf8ToString());
        Term aux = new Term("contents",te.term().utf8ToString());
        DocsEnum de = ar.termDocsEnum(aux);
        while ( ( de.nextDoc())!= DocsEnum.NO_MORE_DOCS ) {
            System.out.println("    docID " +de.docID()+" freq "+de.freq());
            Document doc = reader.document(de.docID());
            System.out.print(doc.get("filename"));
        }
    }
}
```

Alternativa + Costosa

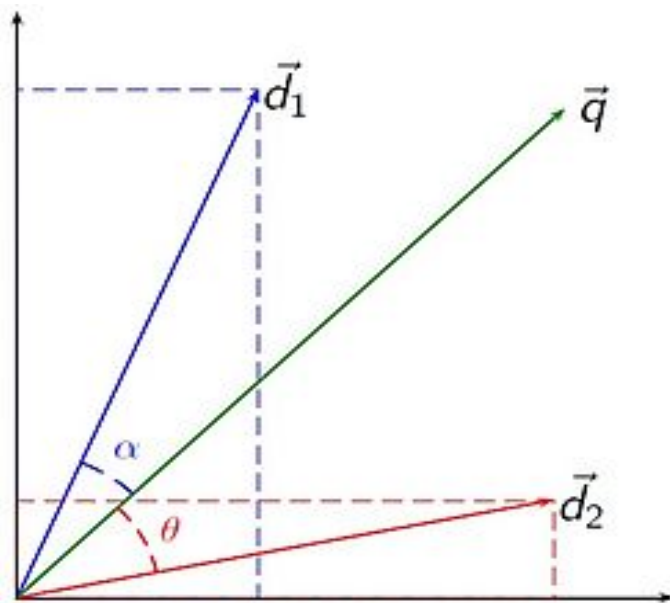
- **Empaquetar** los índices.
 - ♦ Considerar los IndexReaders de cualquier tipo como un AtomicReader, proporcionando acceso a términos, postings, borrado,, valores de doc

IndexReader r;

AtomicReader rd = SlowCompositeReaderWrapper.wrap(r);

- ♦ Hace que un reader compuesto (MultiReader o DirectoryReader) emule AtomicReader. Esto implica implementar la mezcla de postings, términos, etc. sobre la marcha. Internamente utiliza mismos algoritmos que los Readers de Lucene antiguos
- ♦ Fusiona los segmentos

Similarity: Función de Similaridad



$$\text{sim}(d_j, q) = \frac{d_j \cdot q}{\|d_j\| \|q\|} = \frac{\sum_{i=1}^N w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^N w_{i,j}^2} \sqrt{\sum_{i=1}^N w_{i,q}^2}}$$

Similaridad en Lucene

- Modelo conceptual (versión de la medida coseno)

$$\text{score}(q,d) = \text{coord-factor}(q,d) \cdot \text{query-boost}(q) \cdot \frac{V(q) \cdot V(d)}{|V(q)|} \cdot \text{doc-len-norm}(d) \cdot \text{doc-boost}(d)$$

Cómo se calcula en la práctica:

$$\text{score}(q,d) = \text{coord}(q,d) \cdot \text{queryNorm}(q) \cdot \sum_{t \text{ in } q} (\text{tf}(t \text{ in } d) \cdot \text{idf}(t)^2 \cdot t.\text{getBoost}() \cdot \text{norm}(t,d))$$

$$\text{score}(q,d) = \text{coord}(q,d) \cdot \text{queryNorm}(q) \cdot \sum_{t \in q} (\text{tf}(t \text{ in } d) \cdot \text{idf}(t)^2 \cdot t.\text{getBoost}() \cdot \text{norm}(t,d))$$

- $\text{tf}(t \text{ in } d)$: por defecto $\text{frequency}^{1/2}$
- $\text{idf}(t)$: por defecto $\text{idf}(t) = 1 + \log (\text{numDocs} / (\text{docFreq} + 1))$
- $\text{coord}(q,d)$ Depende del cuantos términos de la consulta se encuentran en el documento, p.def: $\text{coord}(q,d) = \#(q \text{ and } d) / \#q$
- $\text{queryNorm}(q)$ factor de normalización para poder comparar entre distintas consultas., def: $\text{queryNorm}(q) = 1 / (q.\text{getBoost}() \cdot \text{sqrt}(\sum_t \text{wtq}^2))$

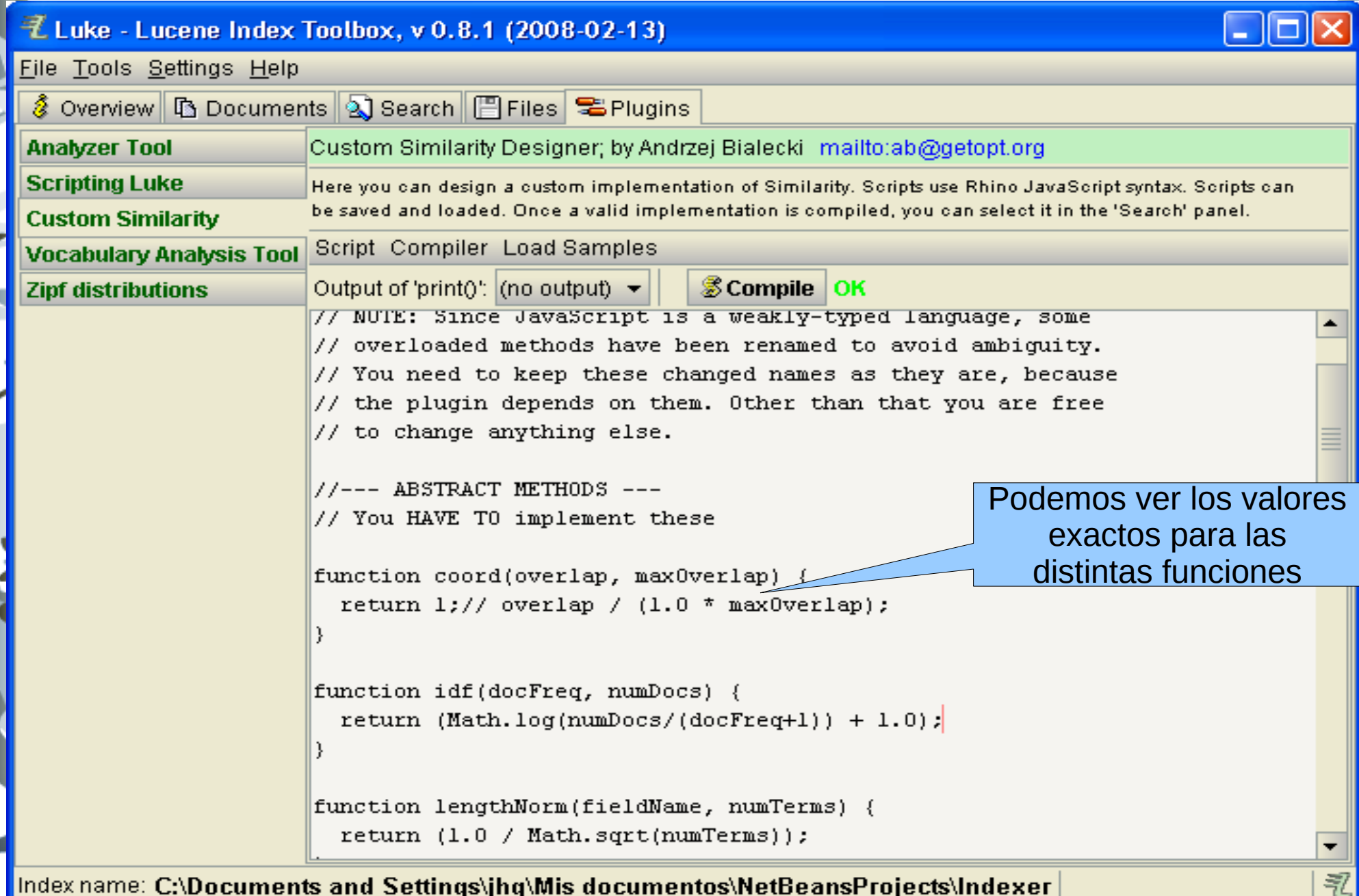
Por ejemplo en el caso de consulta booleana, se computa como

$$\text{wtq} = \text{idf}(t) \cdot t.\text{getBoost}()$$

- $t.\text{getBoost}()$ el boost del término en tiempo de búsqueda
- $\text{norm}(t,d)$ encapsula normalizacion en tiempo de indexación
 - Field boost – boost del campo
 - lengthNorm – se calcula cuando el documento se añade al índice, teniendo en cuenta el numero de tokens en cada campo.

$$\text{norm}(t,d) = \text{lengthNorm}(\text{field}(t) \text{ in } d) \cdot \prod_{\text{field}(t) \text{ in } d} f.\text{boost}()$$

Luke: consultar la función de similitud



Luke - Lucene Index Toolbox, v 0.8.1 (2008-02-13)

File Tools Settings Help

Overview Documents Search Files Plugins

Analyzer Tool

Scripting Luke

Custom Similarity


Vocabulary Analysis Tool

Zipf distributions

Custom Similarity Designer; by Andrzej Bialecki <mailto:ab@getopt.org>

Here you can design a custom implementation of Similarity. Scripts use Rhino JavaScript syntax. Scripts can be saved and loaded. Once a valid implementation is compiled, you can select it in the 'Search' panel.

Script Compiler Load Samples

Output of 'print()': (no output)  **Compile** **OK**

```
// NOTE: Since JavaScript is a weakly-typed language, some
// overloaded methods have been renamed to avoid ambiguity.
// You need to keep these changed names as they are, because
// the plugin depends on them. Other than that you are free
// to change anything else.

//--- ABSTRACT METHODS ---
// You HAVE TO implement these

function coord(overlap, maxOverlap) {
    return 1;// overlap / (1.0 * maxOverlap);
}

function idf(docFreq, numDocs) {
    return (Math.log(numDocs/(docFreq+1)) + 1.0);
}

function lengthNorm(fieldName, numTerms) {
    return (1.0 / Math.sqrt(numTerms));
}
```

Index name: C:\Documents and Settings\jhq\Mis documentos\NetBeansProjects\Indexer

Podemos ver los valores exactos para las distintas funciones

Nuevos Modelos de RI: modificar la funcion de similaridad

- En general `DefaultSimilarity` es suficiente. Pero en algunas aplicaciones podría ser necesario modificar el criterio de similaridad
 - Por ejemplo, no distinguir entre documentos cortos y largos
- La nueva definición debe tenerse en cuenta tanto a la hora de indexar como de buscar.
- Se debe implementar nuestra propia definición de similaridad (extender la clase (o subclases en `Similarity`) y utilizarla llamando a
 - `Searcher.setSimilarity (new Similarity..)`



Escalabilidad:

- Puede indexar unos 95GB/hora con un hardware moderno
- Requiere poca RAM -- sólo 1MB heap
- Indexación incremental es eficiente (tanto como hacerlo en batch)
- Tamaño del índice es aprox. 20-30% del tamaño del texto indexado.

https://lucene.apache.org/core/4_6_0/benchmark/index.html

Escalabilidad

- Query Rate
 - ♦ Lucene es rápido – Utiliza almacenamiento cache
 - ♦ Puede gestionar grandes cargas de trabajo (tiene un crecimiento casi lineal)
 - ♦ Podemos añadir más servidores de consultas
- Index size
 - ♦ Puede manejar fácilmente millones de documentos
 - ♦ Aunque el rendimiento de consulta puede degradarse, añadir documentos al índice tiene un lento factor de crecimiento.
 - ♦ Las principales limitaciones relacionadas con el tamaño del índice las encontramos con la capacidad del disco y los límites de E/S en disco.
 - ♦ Si necesitamos índices mayores,
 - ➔ Disponemos de métodos que permiten realizar consultas sobre varios índices remotos

Lucene



Concurrencia:

- Múltiples buscadores pueden acceder a los índices de Lucene a la vez.
- Los escritores o lectores de los índices de Lucene pueden editarlos mientras se realiza la búsqueda.
- Múltiples escritores o lectores de los índices pueden intentar editar a la vez.

Lucene



- Nutch es un software que implementa un buscador web.
- Está construido sobre Lucene y ofrece funcionalidades específicas como la araña, gestión de enlaces, parsers de HTML, etc.
- Realiza la gestión de temas complicados en los que no queremos programar y gastar tiempo.

Resumen



Document
super_name: Spider-Man
name: Peter Parker
category: superhero
powers: agility, spider-sense

addDocument()

IndexWriter

Consulta
(powers:agility)

Hits
(Matching Docs)

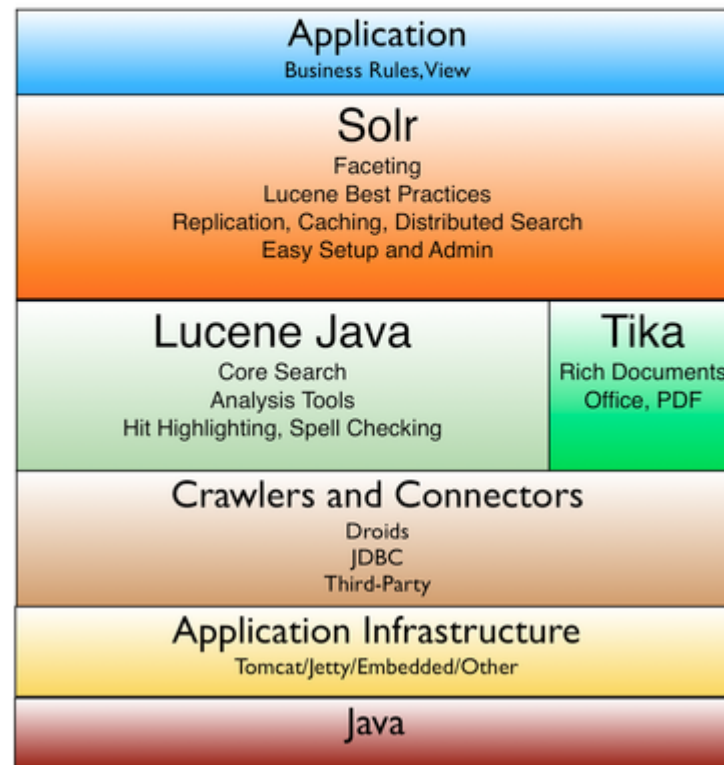
search()

IndexSearcher

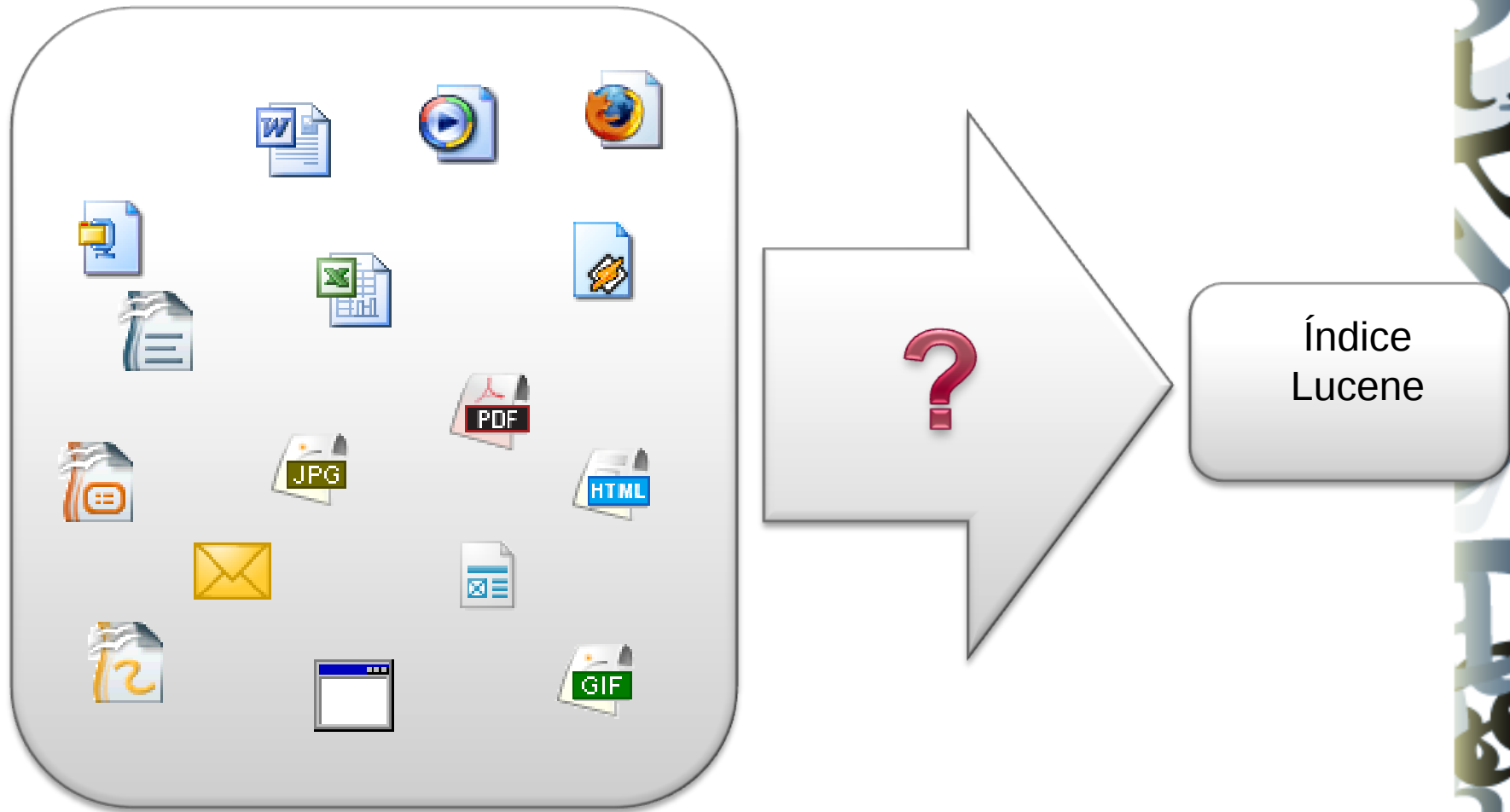
Lucene Index

1. Obtener el .jar de Lucene.
2. Escribir el código de indexación para obtener los datos y crear los objetos Document.
3. Escribir código para crear los objetos de la consulta.
4. Escribir código para utilizar/mostrar resultados.

The Lucene Stack



Gestión de distintos formatos de texto



Analizando otros formatos

- Hasta ahora, sólo hemos considerado ficheros texto (aunque hemos utilizado distintos campos para almacenar la información)
- En una aplicación real, debemos poder trabajar con otros formatos : pdf, html, xml, word, ...

Aunque Lucene no manipula este tipo de documentos, podemos encontrar herramientas que permiten extraer el texto de los documentos y posteriormente indexarlas.

Tika: Analizador de Contenido

Presentación obtenida de
Content analysis with Apache Tika

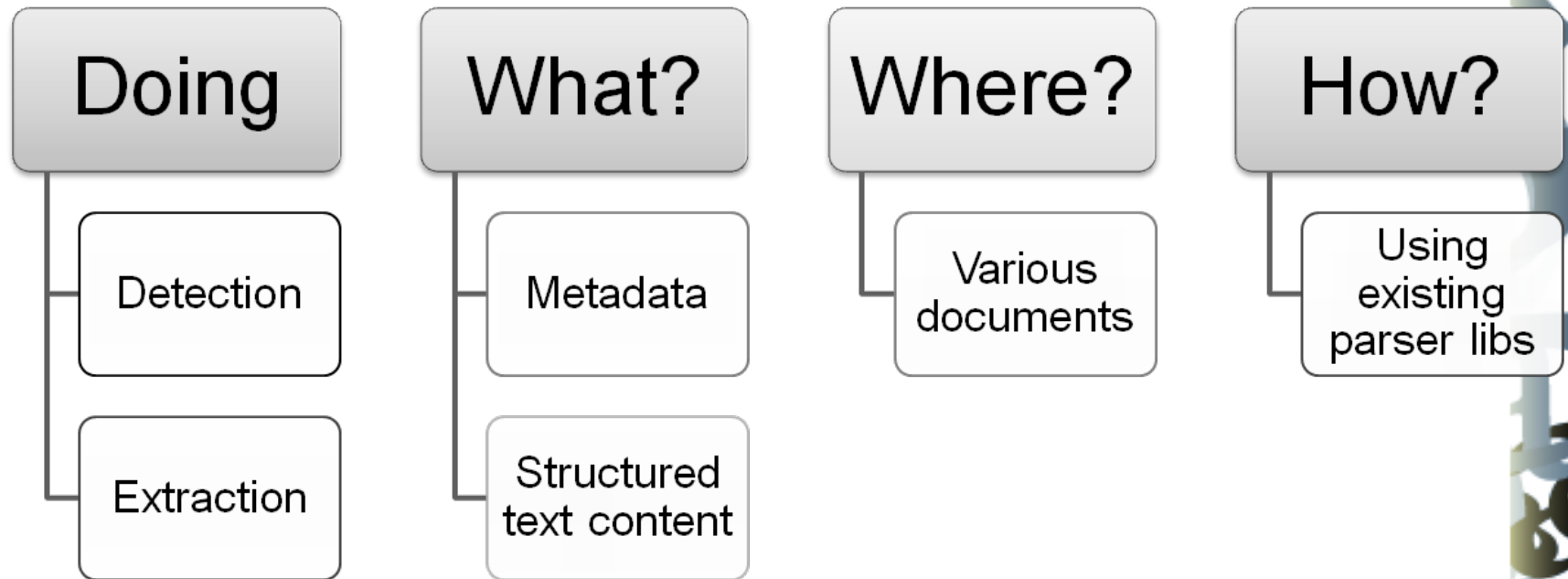
Paolo Mottadelli

Más detalle en

<https://tika.apache.org/1.4/parser.html#apiorgapachetikaparserParser.html>

What is Tika?

It is a Toolkit



Tika Design

The Parser interface

Document input stream

XHTML SAX events

Document metadata

Parser implementations

The Parser interface

```
void parse(InputStream stream, ContentHandler handler,  
Metadata metadata, ParseContext context) throws  
IOException, SAXException, TikaException;
```



Tika Design

The Parser interface

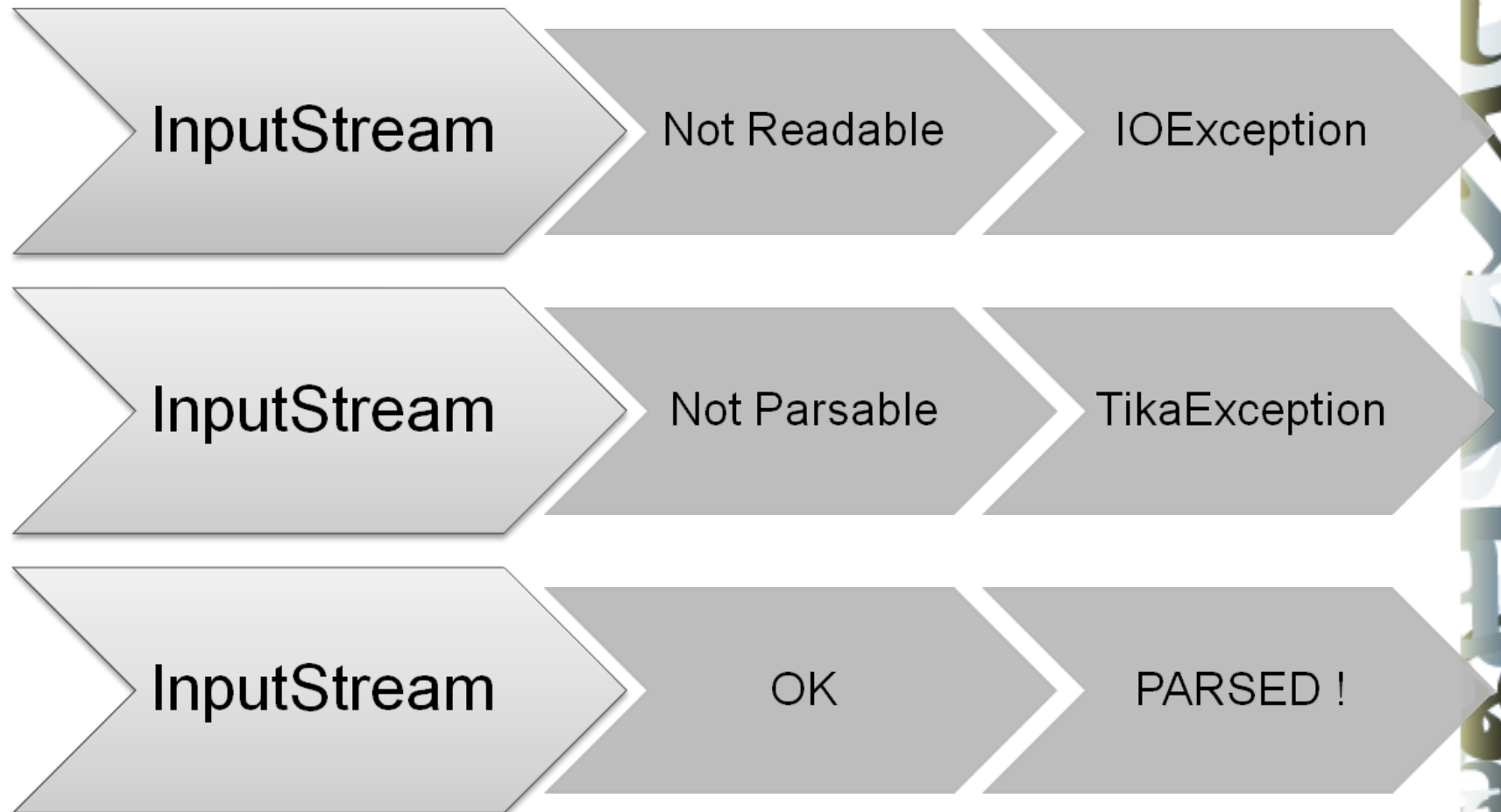
Document input stream

XHTML SAX events

Document metadata

Parser implementations

Document input stream



Tika Design

The Parser interface

Document input stream

XHTML SAX events

Document metadata

Parser implementations

XHTML SAX events

```
<html xmlns="http://www.w3.org/1999/xhtml">  
  <head>  
    <title>...</title>  
  </head>  
  <body> ... </body>  
</html>
```

parse()

SAX events

ContentHandler

Why XHTML?

- Reflect the structured text content of the document
- Not recreating the low level details
- For low level details use low level parser libs

ContentHandler (CH) and Decorators (CHD)

XHTMLContentHandler

- CHD that Produces XHTML events

BodyContentHandler

- CHD that only passes the body to a CH

TextContentHandler

- CHD that only passes characters to a CH

Tika Design

The Parser interface

Document input stream

XHTML SAX events

Document metadata

Parser implementations

Document metadata

Metadata.RESOURCE_NAME_KEY

- The name of the file or resource that contains the document

Metadata.CONTENT_TYPE

- According to the content type the document was parsed to

Metadata.TITLE

- If the document format contains an explicit title field

Metadata.AUTHOR

- If the document format contains an explicit author field

... more metadata: HPSF

Apache POI: HPSF

- **TITLE** - Title
- **SUBJECT** - Subject
- **AUTHOR** - Author
- **KEYWORDS** - Keywords
- **COMMENTS** - Comments
- **TEMPLATE** - Template
- **LAST_SAVED** - Last Saved By
- **REVISION_NUMBER** - Revision Number
- **LAST_PRINTED** - Last Printed
- **LAST_SAVED** - Last Saved Time/Date
- **LAST_SAVED** - Last Saved Time/Date
- **PAGE_COUNT**- Number of Pages
- **WORD_COUNT**- Number of Words
- **CHARACTER_COUNT** - Number of Characters
- **APPLICATION_NAME** - Name of Creating Application

Tika Design

The Parser interface

Document input stream

XHTML SAX events

Document metadata

Parser implementations

Parser implementations

3° Party Libraries

- PDFBox
- Apache POI

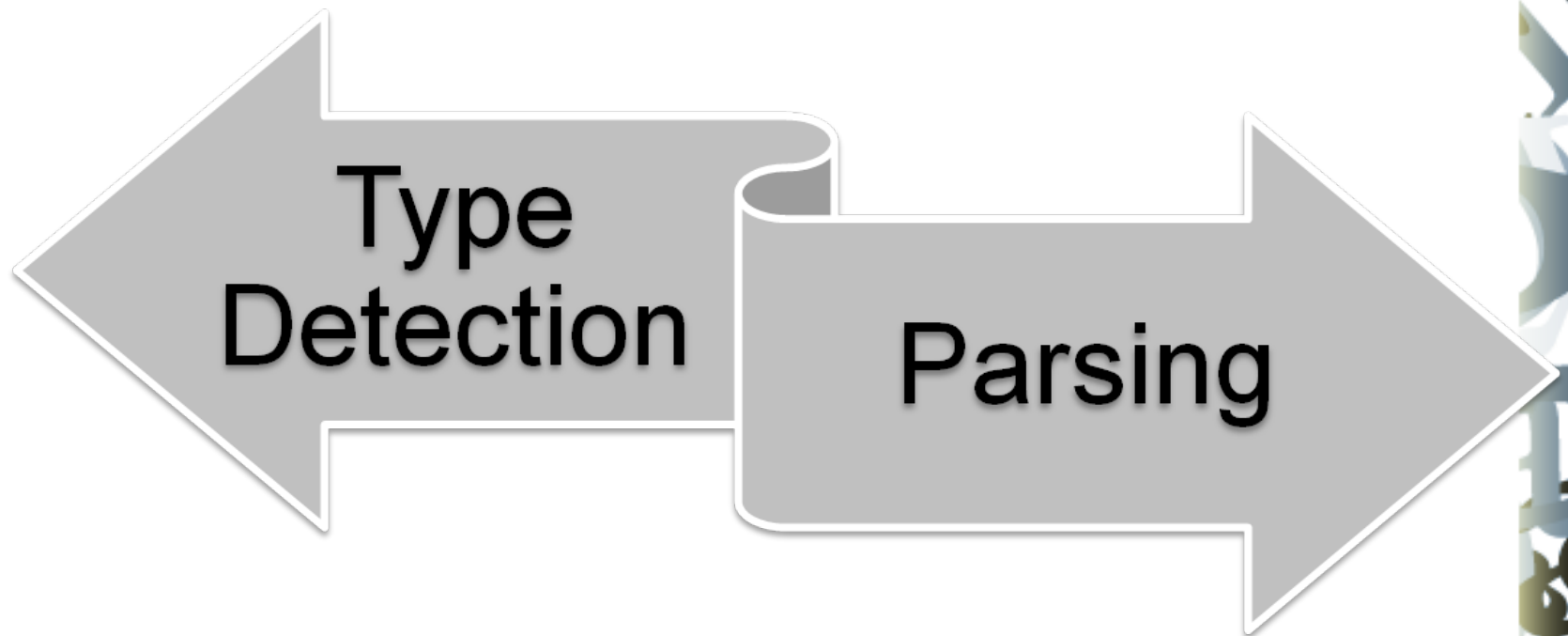


Tika Parsers



















- PDFParser
- OfficeParser

The AutoDetectParser

- Encapsulates all Tika functionalities
- Can handle any type of document



Supported formats

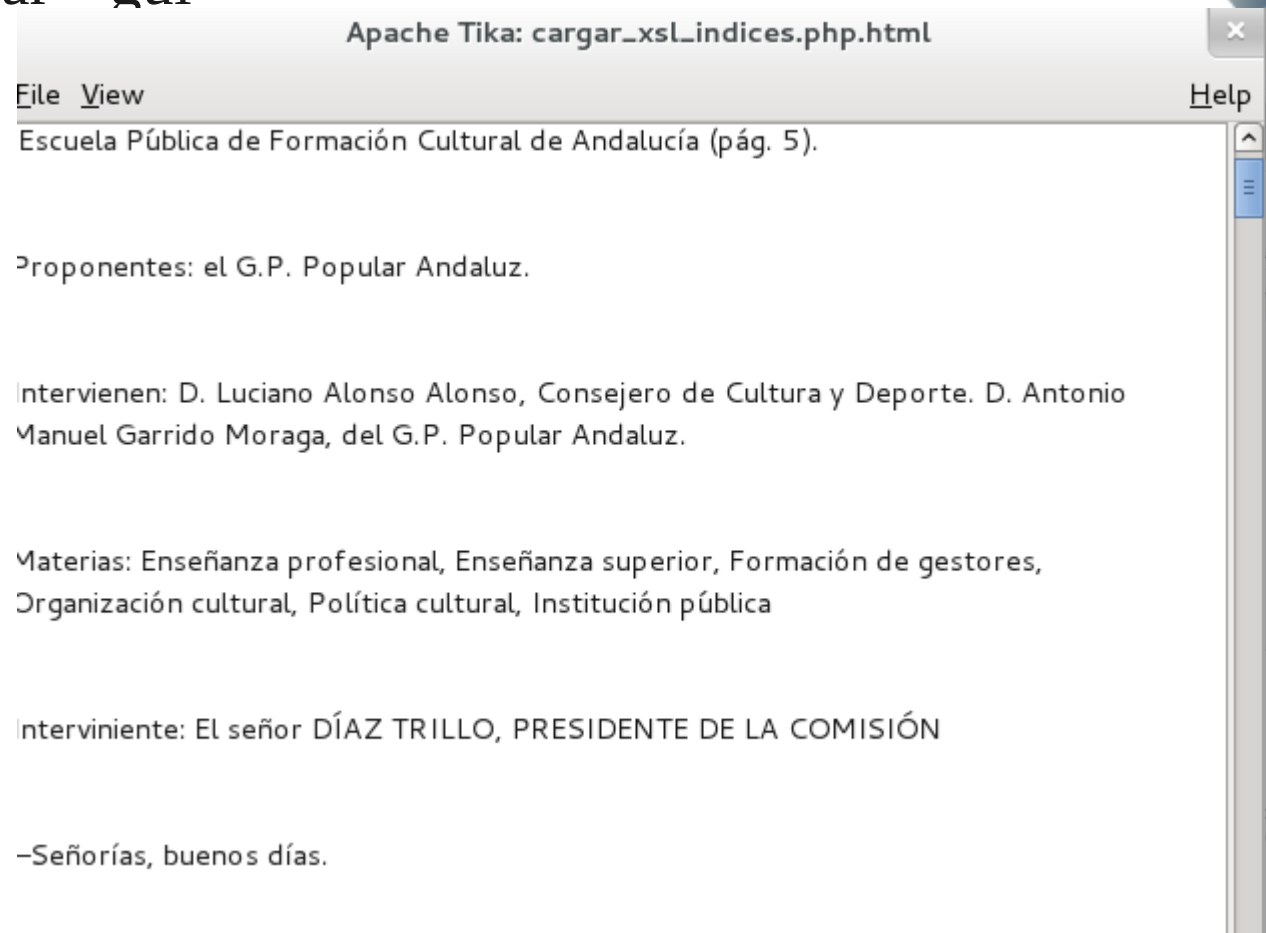
 Excel	 XML
 Word	 HTML
 Power Point	 Images
 PDF	 Java class & jar
 Plain Text	 MP3
 RTF	 OpenDocument
 Outlook	 Tar
 Gzip	 ZIP
 Bzip2	 ...others...

A really simple example

```
InputStream input =  
    MyTest.class.getResourceAsStream("testPPT.ppt");  
  
Metadata metadata = new Metadata();  
ContentHandler handler = new BodyContentHandler();  
  
new OfficeParser().parse(input, handler, metadata);  
  
String contentType = metadata.get(Metadata.CONTENT_TYPE);  
String title= metadata.get(Metadata.TITLE);  
String content = handler.toString();
```

Usando TIKA

- Dispone de una Interfaz de gráfica que nos permite ver lo que podemos extraer de un fichero
- `Java -jar tika-app-1.4.jar --gui`



Tambien directamente desde Lucene HTMLParser

File f

```
Document doc = new Document();  
doc.add(new Field("path", f.getPath().replace(dirSep, '/').....);  
doc.add(new Field("modified",  
    DateTools.timeToString(f.lastModified(),.....);  
doc.add(new Field("uid", uid(f),....);
```

```
FileInputStream fis = new FileInputStream(f);  
HTMLParser parser = new HTMLParser(fis);
```

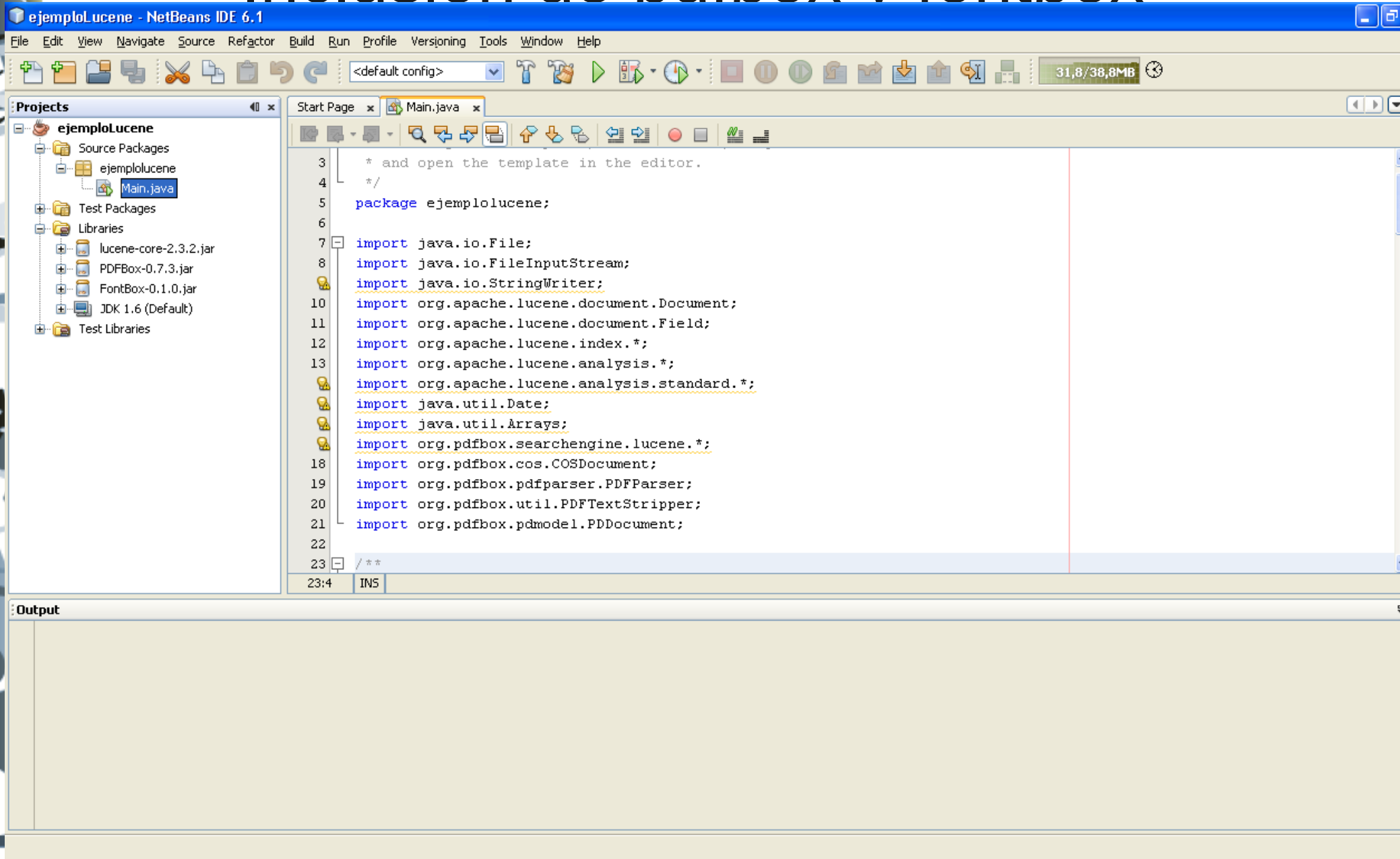
```
doc.add(new Field("contents", parser.getReader()));  
doc.add(new Field("summary", parser.getSummary(),  
    Field.Store.YES, Field.Index.NO));  
doc.add(new Field("title", parser.getTitle(),  
    Field.Store.YES, Field.Index.TOKENIZED));  
return doc;
```

- La mayoría de las páginas webs están en HTML
- Podemos encontrar distintos parsers
 - ♦ Jtidy
 - ♦ NekoHTML
 - ♦

Indexando un PDF

- Portable Document Format (PDF) está ampliamente distribuido
- Permite trabajar con figuras, colores, hiperenlaces, ...
- ¿Cómo podemos trabajar con documentos PDF?
 - ♦ Utilizaremos PDFBox
 - ♦ Lo debemos de incluir en las librerías
 - ♦ También hará falta incluir FontBox

Inclusion de pdfbox v fontbox




PDFBox - Java PDF Library - Mozilla Firefox

Archivo Editar Ver Historial Marcadores Herramientas Ayuda

http://www.pdfbox.org/

Hotmail gratuito Personalizar vínculos Windows Media Windows

PDFBox

Home

About

Index

Download

Nightly Build

Forums

Issues

SourceForge

References

Donations

License

Release Notes

Command Line Utilities

Developers Guide

PDFBox - Java PDF Library

Introduction

Features

Introduction

PDFBox is an open source Java PDF library for working with PDF documents. This project allows creation of new PDF documents, manipulation of existing documents and the ability to extract content from documents. PDFBox also includes several command line utilities.

Features

- PDF to text extraction
- Merge PDF Documents
- PDF Document Encryption/Decryption
- Lucene Search Engine Integration
- Fill in form data FDF and XFDF
- Create a PDF from a text file
- Create images from PDF pages

PDF

Encontrar: custo

Siguiente

Anterior

Resaltar todo

☐ Coincidencia de mayúsculas/minúsculas

Clases de PDFBox

- **PDFParser**

Es la clase que contiene el parser para poder analizar documentos pdf.

Dado un documento PDF, el método **parse()** sera el encargado de parsearlo

```
Import org.pdfbox.pdfparser.PDFParser;
```

- **COSDocument**

Es la representacion en memoria del documento pdf

Es necesario cerrar (close()) el objeto cuando hayamos finalizado de utilizarlo.

```
Import org.pdfbox.cos.COSDocument;
```

- **PDDocument**

Clases de PDFBox

una representación en memoria (a mas alto nivel) del documento pdf.

Para ser valido, un documento debe tener al menos una pagina.

```
import org.pdfbox.pdmodel.PDDocument;
```

- **PDFTextStripper**

Esta clase toma un documento PDF y extrae todo el texto, pasando por alto su formato.

Es responsabilidad de los usuarios de la clase verificar que un determinado usuario tiene los permisos correctos para extraer el texto de documento PDF.

- **Import org.pdfbox.util.PDFTextStripper**

Clases de PDFBox

```
PDFParser parser = new PDFParser(fis);
parser.parse();
COSDocument cos = parser.getDocument();
PDDocument pdd = new PDDocument(cos);
if (pdd.isEncrypted()) {
    System.out.println("Error");
} else {
    System.out.println("No encriptado");
}

String cont_texto = null;
PDFTextStripper stripper = new PDFTextStripper();
String texto = stripper.getText(pdd);
System.out.println("Texto:"+texto);
Document lucenedoc= new Document();
lucenedoc.add(new
Field("contenido",texto,Field.Store.NO,Field.Index.TOKENIZED));
```