Analyse mit Lucene

Dr. Christian Herta

Mai, 2009

Lernziele - Inhalt

- Analyse-Prozess
- Einfluss der Analyse auf die Indizierung und Suche über `QueryParser`
- wichtigen Klassen und Methoden zur Analyse

Outline

Was ist Analyse?

- Wiederholung: Terme
  - Einträge im *Dictionary*
  - zu den Terme gehört auch Feldangabe
  - zu den Termen gibt es *Posting-Listen*
- Analyse: Konvertierung der Feldinhalte in Terme
  - Tokenisierung
  - Filtern und Normalisierung

Abhängigkeit der Analyse(schritte)

- Anwendungs- und Domänenabhängig
  - Großschreibung
  - Terminologie der Domäne: Zahlen (wie Artikelnummern), Emails
  - Sonderzeichen1
- Sprachabhängig
  - Char-Encoding
  - Stemming und Lemmatisierung
  - Stoppworte
  - Akzente

Analyzer

- (abstakte) Klasse, die den Analyse-Prozess kapselt
- in den Unterklassen wird die Analyse implementiert
- Anwendung bei
  - Indizierung
  - auf den Fragestring (im `QueryParser`)

Wesentliche Aufgabe des `Analyser`

In einem Analyser wird eine Analysekette bestehend aus einem Tokenizer und anschließenden Tokenfiltern definiert

Wichtige Implementierungen

- `WhitespaceAnalyser`: lediglich Splitten der Token an *whitespaces*
- `SimpleAnalyser`: *Token-Split* an *non-letter chars*; Wegwerfen von numerischen *chars*
- `StopAnalyser`: wie `SimpleAnalyser`; zusätzlich Stoppwort-Filter (*default:* eine englische Liste)
- `StandardAnalyser`: *lowercasing*, Stopwort-Filtern, Erkennen von Host-Namen, Email Addressen etc. via Parsen
- In der Sandbox `Analyser` für andere Sprachen, wie `GermanAnalyser`

Recap: Analyse bei der Indizierung

- Ob analysiert wird, kann bei der Felderzeugung eingestellt werden
- Welcher `Analyzer` verwendet wird, wird im `IndexWriter`-Konstruktor angegeben
- Welcher `Analyzer` verwendet wird, kann auch auf Dokument-Ebene angegeben werden

Analyse im QueryParser

QueryParser verwendet für standardmäßig für **alle** Felder den vorgegebenen Analyser.

Achtung

Auch bei Felder die bei der Indizierung nicht analysiert wurden, wird der Analyser verwendet ◂ Lösung später

Achtung

Der Analyzer des QueryParser muss zum Analyzer der Indizierung passen

Outline

Implemetierung der Analyser

- konkreter `Analyser` muss (nur) `TokenStream`-Methode implementieren
- kann `reuseableTokenStream`-Methode implementieren - für bessere Performance

Beispiel für einen Analyzer

Listing 1: Tokenizer und zwei Filter

```
1 public final class ExampleAnalyzer extends Analyzer {
2     public TokenStream tokenStream(String fieldName, Reader reader){
3         TokenStream result = new StandardTokenizer(reader);
4         result = new LowerCaseFilter(result);
5         return new StopFilter(result, StandardAnalyser.STOP_WORDS);
6     }
7 }
```

Ausgabe des Analyzer: *Stream of Token*

- *Token* bestehen aus
  - Text-Wert
  - Meta-Daten
- Recap: Werden die *Token* in den Index geschrieben, heißen sie *Terme*

Meta-Daten eines Token

- *Offset-Informationen*: Start- und Endoffset
  - *Character Offsets* im Orginaltext
  - z.B. können für *Highlighting* in `Termvektoren` gespeichert werden
- *positionIncrement*
  - relative (Token-)Position im Vergleich zum vorherigen Token des Tokenstreams
  - für alle eingebauten Tokenizer *Default*-Wert 1
  - für Phrasesuche (und *Span-Queries*)
  - Beseitigung von Stopworten könnten Lücken erzeugen
  - Synonyme kann man auch die gleiche Position setzten
- *token type*
  - z.B. *Email*
  - werden nicht indiziert

Beispiel: Token Meta-Daten mit StandardAnalyser

```
Text: "I'll not respond if you write
to spam@christian.herta.de"

1: [i'll:0->4:<APOSTROPHE>]
2: [respond:9->16:<ALPHANUM>]
3: [you:21->24:<ALPHANUM>]
4: [write:25->30:<ALPHANUM>]
5: [spam@christian.herta.de:34->56:<EMAIL>]
```

Position: [term:startOffset->endOffset:type]

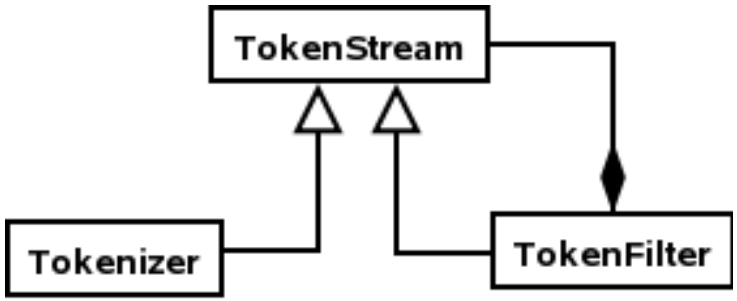
TokenStream

- Zwei prinzipielle Arten:
- `Tokenizer`:
  - arbeitet auf char-Ebene
  - *Input* `Reader`, z.B. `StringReader`
- `Tokenfilter`
  - arbeitet auf Token-Ebene
  - wird vom `Tokenizer` oder anderen `Tokenfilter` "gefüttert"

Softwaredesign Tokenstream

Composite-Pattern

- Tokenizer und TokenFilter sind abstrakte Unterklassen von TokenStream
- TokenStream ist Teil von TokenFilter



Methoden der (abstrakten) (Basis-)Klasse TokenStream

Achtung

Tokenstream-API ändert sich mit Version 2.9 - Tokenstream wird Unterklasse von AttributeSource - keine Token-Objekte

- `Token next(Token reuseableToken)`: gibt nächstes Token oder `NULL` bei *EOS* zurück - Lucene 2.4.1 und älter
- `boolean incrementToken()`: springt zu nächstem Token; Zugriff auf Inhalt über Attribute - ab Lucene 2.9
- `void reset()`: *reset* des Streams zum Anfang
- `void close()`: assoziierte Resoucen des Streams freigeben

Tokenizer

- *Input* Reader, z.B. StringReader
- viele Tokenizer-Implementierungen in Lucene Core-API verfügbar: wie CharTokenizer, WhitespaceTokenizer, KeyTokenizer, LetterTokenizer, LowerCaseTokenizer, SinkTokenizer, StandardTokenizer etc.
- StandardTokenizer nutzt Grammatik über *JFlex* und nutzt *TypeAttribute*: Erkennen von Email (type: EMAIL), Termen mit Apostrophen (wie *I'll*)(type: APOSTROPHE),

TokenFilter

- *Input* TokenStream, d.h. Tokenizer oder anderer TokenStream, so können Filterketten definiert werden
- viele Tokenizer-Implementierungen in Lucene Core-API verfügbar: wie LowerCaseFilter, StopFilter, PorterStemFilter, TeeTokenFilter, ISOLatin1AccentFilter, CachingTokenFilter, LengthFilter, StandardFilter

Splitten mittels TeeTokenFilter

- Splittet den TokenStream in zwei auf:
  - Einer füttert einen SinkTokenizer, welcher den Stream *cached*;
  - Andere normaler Tokenizer-Output
- Anwendung: zwei oder mehr Felder beginnen mit der gleichen Analyse

Token Attribute

Attribut	Beschreibung
TermAttribute	Text des Tokens
PositionIncrementAttribute	(Token) Positions-Inkrement
OffsetAttribute	Start- und End-Char Offset
TypeAttribute	Typ des Tokens
FlagsAttribute	Bits zur Codierung von Flags
PayloadAttribute	byte[] Payload

Eigene Attribute möglich: Unterklassen von `Attribute`

Neuere Luceneversionen - ab 2.9

`TokenStream` erzeugt keine neuen Token bei der Iteration über den Stream, sondern setzt die Attribute neu

Reihenfolge der Analyseschritte

Die Reihenfolger der Analyseschritte ist wichtig

Beispiel

Hat man eine Stopwortliste mit klein geschriebenen Stoppworten so muss *Lowercasing* vorher, nicht nachher geschehen.

Multi-Valued Fields

- *recap*: Felder können mit unterschiedlichem Inhalt mehrmals zu einem `Document` hinzugefügt werden.
- *position increment gap* ist standardmäßig 0 zwischen Feldern; d.h. *Phrasen-* und *Span-Queries* könnten zwischen Feldern *matchen*
- Methode `analyzer.setPositionIncrementGap(..)` setzt größeres *gap*, um das zu verhindern

Feldspezifische Analyse

- Unterschiedliche `Analyzer` für verschiedene Felder möglich, da zu implementierender Methode `tokenStream(..)` Feld-Name übergeben wird:
  - `TokenStream tokenStream(String fieldName, Reader reader)`
- Anwendungsabhängige Analyser können so für die verschiedenen Felder unterschiedliche TokenStreams setzten (Fallunterscheidung über `fieldName`; z.B. `switch-case` oder `if`)
- Anwendungsunabhängige, eingebauten `Analyzer` können über *utility class* `PerFieldAnalyzerWrapper` feldabhängig genutzt werden

Unanalysierte Felder für die Suche

Lösung: Query-Parser und Analyse

Felder, die bei der Indizierung nicht analysiert wurden, sollen nicht bei der Suche analysiert werden siehe

- Separate Felder im User-Interface zusätzlich zur Freifeldsuche; nur für die Freifeldsuche `QueryParser` benutzen
- Domain-spezifischer Analyzer
- Eigene Unterklasse von `QueryParser`
- `PerFieldAnalyzerWrapper`

Stoppworte und Positionen

- `StopAnalyser` erzeugt keine Lücken, d.h folgende Passagen werden äquivalent (Stoppworte: *is*, *not*)
  - "one is not enough"
  - "one enough"
  - "one is enough"
- durch Methode `setEnablePositionIncrements(true)` des `StopFilters` werden Lücken erzeugt - kann über eigenen `Analyzer` genutzt werden

Beispiel: Token Meta-Daten

```
Text: "I'll not respond if you write
to spam@christian.herta.de"

1: [i'll:0->4:<APOSTROPHE>]
3: [respond:9->16:<ALPHANUM>]
5: [you:21->24:<ALPHANUM>]
6: [write:25->30:<ALPHANUM>]
8: [spam@christian.herta.de:34->56:<EMAIL>]
```

Position: [term:startOffset->endOffset:type]

Stoppworte, Phrasenquery und QueryParser

- Um über den `QueryParser` Dokumente zu finden, die nur Phrasen entsprechen, in der Stoppworte (in der Mitte) enthalten sind, ist Folgendes zu beachten (vgl. Übung):
  - Benutzen eines `Analysers` (bei der Indizierung und Suche) der Lücken lässt
  - Im Queryparser muss zusätzlich `parser.setEnablePositionIncrements(true)` gesetzt werden
  - Die Stoppworte können allerdings beliebig ausgetauscht werden, ohne dass sich die Treffer ändern.

Merke

Nur Termen die indiziert werden, können bei der Suche genutzt werden

Outline

Stemming deutscher Texte mit Lucene

- In der Lucene Sandbox sind zwei `Analyzer` enthalten, mit der deutsche Text zur Indizierung analysiert werden können:
- `GermanAnalyzer`
- `SnowballAnalyzer`

GermanAnalyzer

- Filterkette:
  - `StandardTokenizer`: Grammatik basiert (JFlex), erhält z.B. Punkte in Akronymen
  - `StandardFilter`: entfernt z.B. Punkte aus Akronymen
  - `LowercaseFilter`
  - `StopFilter`: Standardliste ersetzbar
  - `GermanStemFilter`:
    - Suffix-Stripping Algorithmus basierend auf "A fast and simple stemming algorithm for german words" von Jörg Caumann; alle Wörter werden auf folgende sieben Grundendungen reduziert: e, s, n, t, em, er, nd
    - Liste von Wörtern die nicht gestemmt werden soll, kann angegeben werden(*exclusion list*)

Snowball

- *String Processing Language* von Dr. Porter
- Verschiedene Regeln für die einzelnen Sprachen: z.B. Dänisch, Nierderländisch, Finnisch, etc.
- zwei `SnowballAnalyzer`-Varianten für Deutsch:
  - German
  - German2
- `Analyzer = new SnowballAnalyzer("German");`

Lemmatisierung

Recap: Begriffsklärung Lemmatisierung

Unter Lemmantisierung versteht man die lexikographische Reduktion eines Wortes auf seine linguistische Grundform

- Lemmatisierung liefert gerade für Deutsch bessere Ergebnisse als (einfaches) Stemming
  - z.B. Häuser: Lemma Haus
- Eine Lemmatisierung ist nicht Teil des Lucene Projektes; Eigener *Lemmafilter* z.B. mit Hilfe kommerzieller Bibliotheken möglich - oder extern bei der Dokumentenverarbeitung

Zusammenfassung

- Analyseprozess von Indizierung uns Suche muss zusammenpassen
- Wahl der Analyseschritte hängt von der Anwendung ab