

# Lucene in a Nutshell



Wie erstelle ich eine Suchanwendung mit Hilfe  
von Lucene ?

Bonsai-Tagung vom 17.11. -18.11.08  
Conni Poppe

# Problemstellung

- Suchmöglichkeiten in eigenen Anwendungen
  - eigene Suchmaschine oder existierende Lösungen
  - Vielfalt der Formate der zu durchsuchenden Informationen
  - Programmieraufwand bei Implementierung und Einbinden der Suchmaschine
  - Suche in Kategorien vs. Volltextsuche
  - Geschwindigkeit der Suchmaschine

# Ziel

- Erstellen eines Indexes und einer Beispielsuche mit Hilfe der Lucene-Bibliothek
  - Erklären der Grundkonzepte und prinzipiellen Vorgehensweise beim Indexieren mit Lucene
  - Implementierung einer Volltextsuche mit JSP
- nicht näher betrachtet:
  - Besonderheiten wie Synonyme, verschiedene Analyser, 'unscharfe Suche', Phrasensuche
- weiterführende Literatur

# Was ist Lucene ?

- Baukasten für eine eigene Suchmaschine
- hochperformante Java-Bibliothek zur Erstellung textbasierter Suchmaschinenanwendungen
- bietet Klassen und Funktionen zur Implementierung von Volltextsuche und Suche in Kategorien
- frei verfügbar beim Apache-Project (OpenSource)
- plattformunabhängig, da vollständig in Java
- kein Crawler bzw. Google-Analyse-Tool (Nutch)
- keine fertige Anwendung

# Benötigte Software

- Java-SDK ab Version 1.5 ( <http://java.sun.com> )
- Lucene-Bibliothek ( <http://lucene.apache.org> )
- Ausführungsumgebung für Java-Code auf dem Server wie Apache-Tomcat zur Bereitstellung der Suchanwendung - bei Bedarf
- GUI wie Eclipse bzw. NetBeans vereinfachen die Javaentwicklung durch einfache Konfiguration der Parameter, Syntax-Highlighting und komfortable Debugfunktionen

# So einfach kann Installation sein

- gehe auf die Lucene-Homepage:  
<http://www.apache.org/dyn/closer.cgi/lucene/java/>
- wähle einen Mirror
- lade die Datei im gewünschten Format (lucene-2.4.0.tar.gz oder lucene-2.4.0.zip)
- nach Auspacken sind das jar-File, die Dokumentation und einige Beispiele verfügbar
- Programme, die Lucene benutzen, müssen die .jar-Datei über den CLASSPATH einbinden

# Strukturen für die Suche

- Klasse 'Document' ist logischer Container für die Indizierung durch Lucene
- 'Document' enthält beliebige Anzahl von Feldern
- 'Field' besteht aus einem Namen (vom Typ String) und einem Wert (String, Date, Reader)
- Struktur eignet sich zum Verwalten von Texten und Metadaten
- Index und Metadaten sind in getrennten Feldern gespeichert

# Indexierung

- Prozess des Vorbereitens und Hinzufügens von Text zu Lucene
  - Optimiert für die Suche
- Lucene indiziert nur Text, d.h.
  - Lucene kümmert sich nicht um Word, PDF, XML, ...
  - wir müssen beliebige Dateiformate, die wir verwenden wollen, in etwas umwandeln, das Lucene indizieren kann
  - es gibt diverse gute OpenSource-Extrahierer

# Aufbau des Index (1)

- Lucene unterscheidet drei Operationen:
  1. Zerlegung in Teile (Tokenizing)
  2. Speichern (Storing)
  3. Indizierung (Indexing)
- nicht jede Kombination der drei Schritte sinnvoll
  - z.B. Zerlegung eines Textes in Teile ohne anschließende Indizierung oder
  - Indizierung ohne Speicherung oder Zerlegung

# Aufbau des Index (2)

- genutzte Kombinationen
  1. Zerlegung der Rohdaten, Indizierung und Speichern als Ganzes ->Text durchsuchen und ausgeben
    - Originaldaten nicht mehr erforderlich
    - großer Index
  2. Text wird zerlegt und indiziert, aber nicht gespeichert
    - Metadaten erforderlich, um das Original wiederzufinden und entsprechende Daten anzuzeigen
    - Pfad oder URL

# Schwachpunkte der Index-API

- Namensgebung und Struktur der Klassen gewöhnungsbedürftig
  - statische Methoden zur Erzeugung der Felder (keine Konstruktoren), Namen nicht selbsterklärend
- kein Satz standardisierter Felder vorhanden
  - Namen frei wählbar: Inhalt vs. Content vs. Contents
  - Interoperabilität zwischen verschiedenen Lucene-Anwendungen nicht automatisch gegeben
- komplexere Daten müssen vorverarbeitet werden, keine Unterstützung durch Lucene-API

# Der Weg in den Index (1)

- Klasse IndexWriter erzeugt und ändert Indizes
- Index im Normalfall Verzeichnis mit Dateien
- dies wird dem IndexWriter als Parameter übergeben
- weitere Parameter von IndexWriter:
  - Analyzer (bestimmt Art der Zerlegung des Textes)
  - Flag, ob Index neu erzeugt oder geändert wird

# Der Weg in den Index (2)

- wichtigste Methode von IndexWriter ist addDocument (Document doc)
- üblicherweise Indizierung über viele Dateien
  - für jede Datei muß ein Dokument mit den gewünschten Feldern erzeugt und dem IndexWriter übergeben werden
- am Ende Aufruf der Methode optimize(), close()
  - Reorganisation des Index und Freigabe der Ressourcen des IndexWriters

# Beim Analytiker (1)

- analysiert den zu durchsuchenden Text und legt fest, was als Suchwort verwendet werden kann
- zerlegt mit Tokenizern die Eingabedaten nach bestimmten Kriterien zu Token (Leerzeichen)
- Tokenfilter bereiten die Token für effiziente Indizierung auf
  - Entfernung von Füllwörtern wie Artikel, Pronomen
  - sprachabhängig: Analyser für verschiedene Sprachen
  - Verarbeitung von Wortstämmen
  - Groß- /Kleinschreibung + Sonderzeichen

# Beim Analytiker (2)

- erhöht die Genauigkeit der Suchergebnisse und die Geschwindigkeit bei der Suche
- Lucene enthält verschiedene Analyser, Tokenizer und TokenFilter
- oft eingesetzte Analyser:
  - StandardAnalyzer
  - WhitespaceAnalyzer
  - SimpleAnalyzer
- eigene Analyser können implementiert werden

# Indizierung zusammengefasst (1)

- Erzeuge einen IndexWriter
- Für jede zu indizierende Datei
  - erzeuge ein Dokument
  - füge die gewünschten Felder zum Dokument hinzu
  - übergib das Dokument an den Indexwriter
- Optimiere den Index (optional)
- Schließe den IndexWriter

# Indizierung zusammengefasst (2)

- Für jedes Dokument:
- Für jedes zu zerlegende Feld:
  - erzeuge Token mit dem spezifischen Tokenizer (Token bestehen aus String, Position, Typ und Offset-Information)
  - übergib die Token bei Bedarf an verkettete TokenFilter, die die Token ändern oder entfernen können
  - Füge das Endergebnis dem Index hinzu

# Listing 1: Indizierung

```
public class IndexFiles {
public static void main(String[] args) {
    ...
    IndexWriter writer=new IndexWriter(idxDir,
        new StandardAnalyzer(), true);
    indexDocs(writer, docDir);
    writer.optimize(),
    writer.close();
}
}
```

## Listing 2: Indizierung

```
static void indexDocs(IndexWriter wr, File f)
{
    if (f.isDirectory()) {
        String files[] = f.list();
        for (int i=0; i <files.length();i++) {
            indexDocs(wr,new File(f, files[i]));
        }
    } else {
        wr.addDocument(createDocument(f));
    }
}
```

# Listing 3: Erstellen eines Lucene-Dokuments

```
public Document createDocument(File f) {  
    Document doc = new Document();  
    doc.add(Field.Text("path", f.getPath()));  
    doc.add(Field.UnIndexed("type", "sdoc"));  
    doc.add(Field.Text("content", new  
        FileReader(f)));  
    return doc;  
}
```

# Verwendung von Luke

- GUI zur Anzeige und Suche auf dem Index

Index name: /Users/grantingersoll/projects/apachecon/europe2007/index  
Number of fields: 3  
Number of documents: 21578  
Number of terms: 103205  
Has deletions?: No  
Index version: 1172436418142  
Last modified: Sun Feb 25 15:51:12 EST 2007  
Directory implementation: org.apache.lucene.store.FSDirectory

Select fields from the list below, and press button to view top terms in these fields. No selection means all fields.

Available Fields: <body>, <date>, <title>

Show top terms >>

Number of top terms: 50

Hint: use Shift-Click to select ranges, or Ctrl-Click to select multiple fields (or unselect all).

Top ranking terms. (Right-click for more options)

No	Rank	Field	Text
1	19043	<body>	3
2	18883	<body>	reuter
3	15345	<body>	said
4	8291	<body>	mln
5	7960	<body>	its
6	7651	<body>	dlrs
7	7590	<body>	from
8	6002	<body>	has
9	5961	<body>	pct
10	5846	<body>	year
11	5479	<body>	company
12	5045	<body>	which
13	4724	<body>	inc
14	4416	<body>	corp
15	4221	<body>	would
16	3996	<body>	were
17	3939	<body>	one
18	3938	<body>	us
19	3877	<body>	have

Index name: /Users/grantingersoll/projects/apachecon/europe2007/index

# Flexible Suche

- neben Indizierung deckt Lucene Suche ab
- gesucht wird über einen „Searcher“ und eine „Query“ oder eine ihrer Unterklassen
- Queries können sehr komplex sein, deshalb existiert in Lucene die Klasse „QueryParser“
  - wurde von Lucene-Entwicklern automatisch aus der Query-Grammatik erzeugt
  - QueryParser wandelt Query-String in ein Objekt der Klasse Query um -> siehe Listing 4

# Queries und Suche

- Query-Syntax ist gut dokumentiert
- neben üblichen Operatoren (AND, OR, NOT) unterstützt L. Wildcards (\* , ?) und Fuzzy-Suche
  - Maier~ findet die verschiedenen Schreibweisen
- weitere Suchmöglichkeiten:
  - Distanzsuche: Begriffe nicht direkt nebeneinander
  - Suche in vorgegebenen Feld: title: „teste mich“
  - Bereichssuche: [Huber TO Maier]
- Gewichtung der Suchbegriffe durch Boost (^)

# Listing 4: Suchen im Index

```
public Document[] search(String qs) {
    String dir = iIndexDir;
    Searcher searcher = new IndexSearcher(dir);
    Query query = QueryParser.parse(qs,
        iDefaultField, iAnalyzer);
    Hits hits = searcher.search(query);
    Document[] rs= new Document[hits.length()];
    for (int i =0; i <rs.length; i++)
        rs[i] = hits.doc(i);
    searcher.close();
    return rs;
}
```

# Spezialfälle

- Behandlung von Zahlen und Datumsfeldern
- Ändern der Positionsinformation
- Abstände zwischen zu suchenden Begriffen
- Hinzufügen von Synonymlisten
- Phrasensuche
- 'unscharfe' Suche
- Bereichssuche