

PyLucene

Installation, Verwendung, Probleme, Lösungen

DZUG-Tagung 2010

Stefan Schwarzer, SSchwarzer.com
info@sschwarzer.com

Dresden, Germany, 2010-09-17

Danke!

- Vortrag basiert auf einem PyLucene-Projekt für den Informationsdienst Wissenschaft e. V. (idw)
<http://idw-online.de>
- Code im Vortrag stammt teilweise aus dem Projekt
- Verwendung mit freundlicher Genehmigung

Einführung

Lucene

- Lucene ist ein Suchmaschinen-Framework
- geschrieben in Java
- verwendet ein eigenes Index-Format, also keine weitere Datenbank notwendig
- Prinzip: Dokumente im Index ablegen, später nach passenden Dokumenten suchen
- indizierte Dokumente bestehen aus Schlüssel-Wert-Paaren, Schlüssel immer Strings, Werte fast(?) immer
- sehr vielseitig
 - boolesche Verknüpfungen (und, oder, nicht)
 - Wildcards (`Py*on`)
 - Phrase Search ("**zusammengehörende Wörter**")
 - Fuzzy Search (ähnlich klingende Begriffe)
 - Proximity Search (Wörter „in der Nähe“)
 - Hervorhebung von Treffern

Einführung

Lucene-API

- API sehr umfangreich
- 1342 Java-Dateien, 953 Klassen, davon 83 abstrakt (in aktueller Version 3.0.2)
- zum Vergleich Standard-Bibliothek von Python 2.6.5: 3790 Python-Dateien, ca. 2040 Klassen
- also **nicht** „pythonic“ ;-)
- Lucene-API ansprechbar aus CPython mit Hilfe von JCC
→ **PyLucene**

Einführung

Lucene-API-Beispiel

```
import lucene
lucene.initVM()

# Open index and search for the document with id 23.
index = lucene.NIOFSDirectory(
    lucene.File(u"index_directory"))
term = lucene.Term(u"id", u"23")
term_query = lucene.TermQuery(term)
searcher = lucene.IndexSearcher(index)
# Expect just one document.
top_docs = searcher.search(term_query, 1)
docs = [searcher.doc(score_doc.doc)
        for score_doc in top_docs.scoreDocs]
lucene_document = docs[0]
```

Installation

Als vorbereitetes Paket

- aktuelles Windows-Binary unter `http://code.google.com/p/pylucene-win32-binary`
- auf anderen Plattformen teilweise sehr veraltet
- zum Beispiel in aktuellem Ubuntu-Linux 10.04: PyLucene 2.3.1 (aktuell ist 3.0.2, letzte 2er-Version ist 2.9.3)
- oft Installation aus Source-Code nötig

Installation

Allgemeines Vorgehen

- siehe <http://lucene.apache.org/pylucene/documentation/install.html> und <http://lucene.apache.org/pylucene/jcc/documentation/install.html>
- PyLucene herunterladen von <http://apache.abdaal.com/lucene/pylucene> und auspacken
- **in dieser Reihenfolge ...**
- **JCC bauen und installieren**
benötigt Python, GNU Make und C++-Compiler
- **Lucene bauen und installieren**
benötigt JDK und Ant (Java-Build-Tool)

Installation

JCC bauen und installieren

- `cd pylucene-3.0.2-1/jcc`
- ggf. `setup.py` anpassen, insbesondere den Pfad für die eigene Plattform im Dictionary `JDK`
- `python setup.py build`
- falls eine Fehlermeldung kommt, dass kein Shared-Mode genutzt werden kann, als Workaround Umgebungsvariable `NO_SHARED` auf `1` setzen und Build-Vorgang wiederholen (weitere Diskussion ggf. am Ende des Vortrags)
- mit Root-Rechten: `python setup.py install`

Installation

Lucene bauen und installieren

- `cd ..`
- Makefile anpassen (zwingend notwendig)
 - # Ubuntu Linux 8.10 64-bit, Python 2.5.2, OpenJDK ...
 - #PREFIX_PYTHON=/usr
 - #ANT=ant
 - #PYTHON=\$(PREFIX_PYTHON)/bin/python
 - #JCC=\$(PYTHON) -m jcc --shared
 - #NUM_FILES=2

Installation

Lucene bauen und installieren

- `cd ..`
- Makefile anpassen (zwingend notwendig)
 - # meine Plattform, Python 2.5
 - #PREFIX_PYTHON=/usr
 - #ANT=ant
 - #PYTHON=\$(PREFIX_PYTHON)/bin/python
 - #JCC=\$(PYTHON) -m jcc --shared
 - #NUM_FILES=2

Installation

Lucene bauen und installieren

- `cd ..`
- Makefile anpassen (zwingend notwendig)
meine Plattform, Python 2.5
PREFIX_PYTHON=/usr
ANT=ant
PYTHON=\$(PREFIX_PYTHON)/bin/python
JCC=\$(PYTHON) -m jcc --shared
NUM_FILES=2

Installation

Lucene bauen und installieren

- `cd ..`
- Makefile anpassen (zwingend notwendig)
meine Plattform, Python 2.6
PREFIX_PYTHON=/usr
ANT=ant
PYTHON=\$(PREFIX_PYTHON)/bin/python
JCC=\$(PYTHON) -m jcc.__main__ --shared
NUM_FILES=2

Installation

Lucene bauen und installieren

- `cd ..`
- Makefile anpassen (zwingend notwendig)
meine Plattform, Python 2.7
PREFIX_PYTHON=/usr
ANT=ant
PYTHON=\$(PREFIX_PYTHON)/bin/python
JCC=\$(PYTHON) -m jcc --shared
NUM_FILES=2

Installation

Lucene bauen und installieren

- `cd ..`
- Makefile anpassen (zwingend notwendig)
meine Plattform, Python 2.7
PREFIX_PYTHON=/usr
ANT=ant
PYTHON=\$(PREFIX_PYTHON)/bin/python
JCC=\$(PYTHON) -m jcc --shared # ggf. entfernen
NUM_FILES=2

Installation

Lucene bauen und installieren

- `cd ..`
- Makefile anpassen (zwingend notwendig)
meine Plattform, Python 2.7
PREFIX_PYTHON=/usr
ANT=ant
PYTHON=\$(PREFIX_PYTHON)/bin/python
JCC=\$(PYTHON) -m jcc --shared
NUM_FILES=2
- `gmake`
- mit Root-Rechten: `gmake install`

Verwendung

Allgemeines

- **API-Dokumentation**

`http://lucene.apache.org/java/3_0_2/api/all/index.html`

- **PyLucene-Beispiele**

`http://svn.apache.org/viewvc/lucene/pylucene/trunk/samples/`

- für die Lucene-API **nur Unicode** verwenden, keine Bytestrings
- Lucene-Dokumente bestehen aus Feldern (Feldname, Wert)
- Feldnamen **sind immer Zeichenketten**
- Werte meistens praktischerweise auch (`NumericFields` lassen sich anscheinend mit PyLucene setzen, aber nicht auslesen)

Verwendung

Feldeigenschaften

- Beim Anlegen von Feldern können verschiedene Eigenschaften angegeben werden.
- **Stored**: Das Feld ist in einem Lucene-Dokument enthalten und kann mit diesem aus dem Index geholt werden. Ein Feld kann auch dann in einem Query verwendet werden, wenn es **nicht** „stored“ ist!
- **Analyzed**: Der Feldinhalt wird in einzelne Wörter aufgespalten und die Wörter für eine bestimmte natürliche Sprache normalisiert. Damit wird bspw. auch ein Dokument gefunden, das den Suchbegriff in der Mehrzahlform enthält, auch wenn das Query die Einzahl verwendet.
- **TermVectors**: Diese Eigenschaft muss vorhanden sein, um Text-Auszüge für die Treffer zu erzeugen.

Verwendung

Hilfreiche Konstanten

```
LUCENE_VERSION = lucene.Version.LUCENE_30
```

```
MAX_FIELD_LENGTH = \  
    lucene.IndexWriter.MaxFieldLength(10000)
```

```
STORED_YES = lucene.Field.Store.YES
```

```
STORED_NO = lucene.Field.Store.NO
```

```
ANALYZED_YES = lucene.Field.Index.ANALYZED
```

```
ANALYZED_NO = lucene.Field.Index.NOT_ANALYZED
```

```
TERM_VECTORS_YES = \  
    lucene.Field.TermVector.WITH_POSITIONS_OFFSETS
```

```
TERM_VECTORS_NO = lucene.Field.TermVector.NO
```

```
OCCUR = lucene.BooleanClause.Occur
```

```
QUERYPARSER_OPERATOR = lucene.QueryParser.Operator
```

Verwendung

Hilfreiche Funktionen

```
def u(data):
    """Return unicode string for 'data'."""
    if data is None:
        return u""
    else:
        return unicode(data)

def term_query(field, value):
    """Return Lucene term query."""
    return lucene.TermQuery(lucene.Term(field, value))
```

Verwendung

Dokument-Ids

- Lucene vergibt Nummern für Dokumente im Index
- sie sind aber **nicht** fest zugeordnet
- bspw. können sich diese Nummern durch Löschen anderer Dokumente ändern
- daher immer ein Feld mit einer eindeutigen Id mit dem Dokument abspeichern

Verwendung

Wichtige Klassen für die Index-Verwaltung

```
# Generate an index directory on disk.
index = lucene.NIOFSDirectory(
    lucene.File(u"index_directory"))

# You can use 'StandardAnalyzer' for English texts.
analyzer = lucene.GermanAnalyzer(LUCENE_VERSION)

# Index writer, needed for all index changes
writer = lucene.IndexWriter(index, analyzer,
    MAX_FIELD_LENGTH)

# Index searcher, needed for all searches
searcher = lucene.IndexSearcher(index)
```

Verwendung

Dokument hinzufügen

```
lucene_field_data = [  
    (u"id", u"17", STORED_YES,  
     ANALYZED_NO, TERM_VECTORS_NO),  
    (u"firstname", u"Stefan", STORED_NO,  
     ANALYZED_NO, TERM_VECTORS_YES),  
    (u"lastname", u"Schwarzer", STORED_NO,  
     ANALYZED_NO, TERM_VECTORS_YES)]  
lucene_document = lucene.Document()  
for name, value, stored, analyzed, term_vectors in \  
    lucene_field_data():  
    field = lucene.Field(name, value, stored, analyzed,  
                        term_vectors)  
    lucene_document.add(field)  
writer.addDocument(lucene_document, analyzer)  
writer.commit()
```

Verwendung

Dokumente löschen

```
# Delete the document with id 17.  
# Note: 'term_query' is defined above.  
query = term_query(u"id", u"17")  
writer.deleteDocuments(query)  
writer.commit()
```

```
# Delete all documents from the index.  
writer.deleteAll()  
writer.commit()
```

Um ein Dokument zu aktualisieren, muss es **gelöscht und neu angelegt** werden.

Verwendung

Queries erzeugen, 1/2

- **empfohlen**: Such-Queries über API baumartig aufbauen
- bspw. boolesches Query

```
and_query = lucene.BooleanQuery()
and_query.add(term_query(u"tagung", u"DZUG"),
              OCCUR.MUST)
and_query.add(term_query(u"ort", u"Dresden"),
              OCCUR.MUST)
```
- außer **MUST** sind noch **SHOULD** und **MUST_NOT** möglich

Verwendung

Queries erzeugen, 2/2

Alternativ kann ein Query aus einem Query-String erzeugt werden.

```
query_string = u"(Python* OR Zope) AND DZUG"  
query_parser = lucene.QueryParser(LUCENE_VERSION,  
                                   u"description", analyzer)  
# No error handling shown here  
query = query_parser.parse(query_string)
```

Verwendung

Dokumente finden

```
# Return document ids for the first 'max_hits' hits
# for the query.
max_hits = 10
top_docs = searcher.search(lucene_query, max_hits)
doc_numbers = [score_doc.doc
                for score_doc in top_docs.scoreDocs]
return [searcher.doc(doc_number)[u"id"]
        for doc_number in doc_numbers]
```

Verwendung

Index aufräumen und schließen

```
# Optimize index.  
writer.optimize()  
writer.commit()  
  
# Close index. Rollback isn't strictly necessary.  
writer.rollback()  
writer.close()  
index.close()
```

Probleme und Lösungen

Zu wenig oder zu viel Speicher

- Wenn der JVM zu viel oder zu wenig Speicher zur Verfügung steht, kann man ihn in `lucene.initVM` einstellen.
- mögliche Parameter sind `initialheap`, `maxheap` und `maxstack`
- daneben lassen sich noch einige andere Parameter in `initVM` einstellen
- siehe <http://lucene.apache.org/pylucene/jcc/documentation/readme.html#api>

Probleme und Lösungen

Lucene-Ausnahmen

- in der Lucene-API ausgelöste Ausnahmen sind Java-Exceptions
- diese erscheinen in Python-Code nur pauschal als **JavaError**
- Notlösung: erkennen der ursprünglichen Ausnahmeklasse durch Inspektion des Anfangs der Fehlermeldung – Beispiel:

try:

```
    query = query_parser.parse(query_string)
except lucene.JavaError, exc:
    java_exception = exc.getJavaException()
    message = unicode(java_exception)
    if message.startswith(u"Cannot parse ") or \
        message.startswith(u"minimumSimilarity"):
        raise ParserError(message)
```

Probleme und Lösungen

Keine Treffer, 1/3

- wenn sich der Feldname im Index und im Query unterscheiden, gibt es keine Fehlermeldung
- Beispiel:

```
# Add a document.
document = lucene.Document()
field = lucene.Field(u"firsttname", u"Stefan")
document.add(field)
writer.addDocument(document, analyzer)
writer.commit()
# Query the index.
term_query = lucene.TermQuery(
    lucene.Term(u"firstt_name", u"Stefan"))
# No hits!
top_docs = searcher.search(term_query, 1)
```

Probleme und Lösungen

Keine Treffer, 2/3

- Empfehlung: Namenskonvention für Feldnamen festlegen
- ... und natürlich einhalten ;-)
- möglichst fehlerunempfindliche Konvention
- zum Beispiel: alles Kleinbuchstaben, keine Unterstriche
- ggf. durch Wrapper erzwingen
- Beispiel: statt `lucene.Field`:

```
def lucene_field(name, *args):  
    match = re.search(ur"^[a-z]+$", name)  
    assert match is not None, \  
        "field name %s contains invalid characters" % \  
        name  
    return lucene.Field(name, *args)
```

Probleme und Lösungen

Keine Treffer, 3/3

- andere Ursache: Dokument ist dem benutzten `IndexSearcher` unbekannt
- `writer.commit()` vergessen?
- oder Dokument wurde **nach** Instanziierung des benutzten `IndexSearchers` hinzugefügt?
- bei Bedarf neuen `IndexSearcher` erzeugen

```
# Ensure we have a searcher which sees the most  
# recent changes in the index.
```

```
if not searcher.getIndexReader().isCurrent():  
    searcher.close()  
    searcher = lucene.IndexSearcher(index)
```

Probleme und Lösungen

TooManyClauses-Exception

- siehe http://wiki.apache.org/lucene-java/LuceneFAQ#Why_am_I_getting_a_TooManyClauses_exception.3F
- statt `TermRangeQuery` `TermRangeFilter` verwenden
- `BooleanQuery.setMaxClauseCount` verwenden
- Genauigkeit von Range-Queries reduzieren

Danke für die Aufmerksamkeit! :-)

Fragen?

Anmerkungen?

Diskussion?