Anwenderhandbuch
regain

Version 1.1

Autor: Til Schneider, www.murfman.de

Inhalt

1.Einführung....3
2.Der Crawler....4
2.1.Der Crawler-Prozeß....4
2.1.1.Durchsuchen eines Dokuments....4
2.1.2.Durchsuchen eines Verzeichnisses....5
2.1.3.Aufnehmen eines Dokuments in den Suchindex....5
2.1.4.Inkrementelle Indizierung....6
2.1.5.Partielle Indizierung....6
2.2.Abhängigkeiten....6
2.3.Die Installation....7
2.4.Die Konfiguration....8
2.5.Der Start....8
2.6.Das Indexverzeichnis....9
2.7.Exkurs: Reguläre Ausdrücke....10
3.Die Suchmaske....12
3.1.Abhängigkeiten....12
3.2.Die Installation....12
3.3.Die Konfiguration....13
3.4.Der Start....14
3.5.Die Syntax für Suchanfragen von Lucene....14
4.Was ist bei anderen Sprachen zu beachten....17
5.Bewertung von Lucene....18
Anhang A: Die XML-Tags der CrawlerConfiguration.xml....19
Anhang B: Die Konfiguration der Präparatoren....28
B.1.Konfiguration des HtmlPreparator....28
Anhang C: Die XML-Tags der SearchConfiguration.xml....30

1. Einführung

Die Lucene-Suche durchsucht einen Webauftritt und/oder ein Verzeichnis nach Dokumenten und stellt diese in einen Suchindex. Über eine Suchmaske kann auf diesem Index gesucht werden.

Ferner kann man die Lucene-Suche dazu nutzen, tote Links zu finden oder einen serverseitigen Cache automatisch zu befüllen.

Die Lucene-Suche teilt sich in zwei getrennte Anwendungen: Den Crawler und die Suchmaske. Der Crawler hat die Aufgabe, den Suchindex zu erstellen. Die Suchmaske führt auf dem fertigen Index Suchen aus und stellt das Ergebnis dar.

2. Der Crawler

Der Crawler ist eine Java-Stand-Alone-Applikation, die auf der Konsole läuft, also ohne Benutzeroberfläche. Er kann damit also auch automatisch gestartet werden, beispielsweise durch einen cron-Job. Über eine XML-Datei sind sehr weitreichende Konfigurationen möglich.

Er verfügt über eine Schwarze und eine Weiße Liste, mit deren Hilfe sich leicht steuern lässt, welche URLs bearbeitet werden sollen und welche nicht.

Sogenannte Präperatoren übernehmen das Auslesen von Text für die verschiedenen Dateiformate. In der Konfiguration lässt sich einstellen, welcher Präperator für welchen Dateityp genutzt werden soll.

2.1. Der Crawler-Prozeß

In diesem Abschnitt wird erläutert, wie der Crawler arbeitet.

Für jede URL, die bearbeitet werden soll, wird ein „CrawlerJob“ erstellt. Dieser legt fest, ob das Dokument nach weiteren URLs durchsucht werden soll und ob es in den Suchindex aufgenommen werden soll.

Alle CrawlerJobs befinden sich in einer Liste, die nach und nach abgearbeitet wird.

Zeigt die URL auf ein Verzeichnis, so werden für alle Dateien und Unterverzeichnisse CrawlerJobs erzeugt. Danach wird direkt mit dem nächsten CrawlerJob fortgefahren.

Zeigt die URL auf ein Dokument, so wird dieses gegebenenfalls nach URLs durchsucht und/oder in den Suchindex aufgenommen.

Falls an irgendeiner Stelle ein Fehler auftritt, dann wird dieser ausgegeben und in eine Fehlerliste eingetragen. Der Crawler-Prozeß wird dabei nicht unterbrochen. Die Fehlerliste wird am Ende des Crawler-Prozesses noch einmal kompakt ausgegeben und in die Datei errors.txt geschrieben (Zu finden im log-Unterverzeichnis des Index). Außerdem gibt der Crawler den Returncode `1` zurück.

2.1.1. Durchsuchen eines Dokuments

Die konfigurierten HTML-Parser-Suchmuster werden auf den Inhalt des Dokuments angewendet. Für jede so gefundene URL wird wie folgt verfahren:

Die URL wird zuerst in eine absolute URL umgewandelt.

Nun wird geprüft, ob diese URL bereits bekannt ist oder ob sie schon einmal ignoriert wurde. Trifft eines von beidem zu, dann wird die URL verworfen.

Dann wird anhand der Weißen und Schwarzen Liste entschieden, ob sie bearbeitet wird oder nicht. Wenn ja, dann wird ein neuer CrawlerJob erstellt und die URL wird in die Liste der bekannten URLs aufgenommen. Wenn nein, dann wird sie verworfen und in die Liste der ignorierten URLs aufgenommen.

2.1.2. Durchsuchen eines Verzeichnisses

Für jede Datei im Verzeichnis wird geprüft, ob es sich um ein Verzeichnis handelt. Wenn ja, dann wird für dieses Unterverzeichnis ein neuer CrawlerJob erstellt.

Handelt es sich um eine Datei, dann geprüft, ob ein Verzeichnis-Parser-Suchmuster zum Dateinamen passt. Wenn ja, dann wird ein neuer CrawlerJob erstellt. Wenn nein, dann wird die Datei verworfen.

2.1.3. Aufnehmen eines Dokuments in den Suchindex

Ein Dokument, das in den Suchindex aufgenommen werden soll, wird zuerst einmal bestimmt, welcher Präperator für das vorliegende Dokumentenformat zuständig ist. Der so ermittelte Präparator befreit nun den Inhalt des Dokuments von seinen Formatierungsinformationen, so dass nur noch der reine zu indizierenden Text übrig bleibt.

Wenn Sie ein weiteres Format unterstützen wollen, so müssen sie einen neuen `Preparator` schreiben und diesen in der Datei `CrawlerConfiguration.xml` in der `<preparatorList>` eintragen.

Der so vorbereitete Text wird nun Lucene übergeben. Lucene analysiert den Text, wobei häufige Worte wie „und“ oder „dass“ herausfallen und Wortendungen abgeschnitten werden, so dass sie plural- und geschlechtsneutral werden. Die so entstandenen Terme werden dem Index hinzugefügt.

Hinweis:

Wenn Sie in der Konfiguration `<writeAnalysisFiles>` auf `true` setzen, dann wird im Index ein Unterverzeichnis mit dem Namen `analysis` angelegt, in dem Analysedateien angelegt werden. Diese bestehen aus einer Datei `AllTerms.txt`, die alle Terme enthält und vielen Dateien, die die jeweiligen Zwischenschritte der Indizierungsvorbereitung beinhalten.

2.1.4. Inkrementelle Indizierung

Wenn es bereits einen alten Index gibt und wenn nicht der Kommandozeilenparameter `–forceNewIndex` angegeben wurde, dann wird eine sog. „Inkrementelle Indizierung“ vorgenommen, es werden also nur geänderte Dokumente indiziert. Dabei wird erst geprüft, ob bereits ein Indexeintrag vorhanden ist und ob dieser aktuell ist. Trifft eine dieser Bedingungen nicht zu, dann wird das Dokument neu indiziert, ansonsten wird der alte Eintrag behalten.

Am Ende des Crawler-Prozesses werden alle Indexeinträge gelöscht, die auf Dokumente zeigen, die es nicht mehr gibt.

2.1.5. Partielle Indizierung

Gerade wenn man sehr große Datenmengen indiziert, ist es manchmal wünschenswert, nur einen Teil des Index zu aktualisieren. Wenn man beispielsweise einen Index über viele große Netzlaufwerke hat, so will man evtl. die Daten auf einer Platte täglich aktualisieren, während der Inhalt einer anderen Platte nur einmal im Monat indiziert werden soll.

Um das zu bewerkstelligen, legt man für jede Platte einen Eintrag in der Whitelist an und gibt ihm mit Hilfe des Attributs `name` einen Namen.

Beispiel:

```
<whitelist>
  <prefix name="Platte-N">file://n:</prefix>
  <prefix name="Platte-M">file://m:</prefix>
  <prefix name="Platte-O">file://o:</prefix>
</whitelist>
```

Um nur die Platten M und O zu aktualisieren ruft man den Crawler wie folgt auf:

```
java –jar regain.jar –onlyEntries Platte-M,Platte-O
```

Achten Sie darauf, zwischen `Platte-M` und `Platte-O` nur ein Komma zu setzen, kein Leerzeichen!

Der Crawler wird nun nur die Inhalte der Platten M und O aktualisieren. Die Indexeinträge für die Platte N werden unverändert im Index belassen.

2.2. Abhängigkeiten

Der Crawler läuft unter Java ab der Version 1.2.2.

Er nutzt die folgenden Bibliotheken:

- Jakarta Regexp 1.3 (`jakarta-regexp-1.3.jar`). Ermöglicht die Nutzung von regulären Ausdrücken. Siehe: http://jakarta.apache.org/regexp/
- Jakarta Log4j 1.2.9 (`log4j-1.2.9.jar`). Stellt die Protokollierung zur Verfügung. Siehe: http://jakarta.apache.org/log4j/
- Jakarta Lucene 1.4.2 (`lucene-1.4.2.jar`). Enthält den Kern der Suche, also Indexerstellung und Suche auf dem Index. Siehe http://jakarta.apache.org/lucene/
- Apache XML Xerces 2.6.2 (`xercesImpl.jar` und `xml-apis.jar`). Bietet einen Parser zum Lesen von XML-Dateien. Siehe http://xml.apache.org/xerces2-j/
- PDFBox 0.6.7 (`PDFBox-0.6.7a.jar`). Ermöglicht das Lesen von PDF-Dokumenten. Läuft nur unter Java 1.3 oder darüber. Siehe http://pdfbox.org/
- Jakarta POI 2.5.1 (`poi-2.5.1-final-20040804.jar` und `poi-scratchpad-2.5.1-final-20040804.jar`). Ermöglicht das Lesen von Microsoft-Excel-Dokumenten und Microsoft-Word-Dokumenten. Das Lesen von Word-Dokumenten ist dabei leider noch in einem sehr frühen Entwicklungsstadium. Siehe http://jakarta.apache.org/poi/
- Jacob 1.8 (`jacob.jar` und `jacob.dll`). Ermöglicht den Zugriff auf COM-Objekte von Java aus. Damit ist das Auslesen von Microsoft Office Dokumenten implementiert, indem die Daten direkt aus den Office-Anwendungen gelesen werden. Siehe `http://sourceforge.net/projects/jacob-project`
- Jacobgen (Verzeichnis jacobgen). Codegenerator für Jacob. Ermöglicht einen einfacheren Zugriff auf COM-Objekte durch die Generierung von Wrapper-Klassen. Die verwendete Version ist eine Weiterentwicklung des STZ-IDA, aufbauend auf der Version 0.3.
  Siehe http://www.bigatti.it/projects/jacobgen/

Hinweis:
Die entsprechenden Jars müssen nicht im Classpath stehen, da sie in das `regain.jar` integriert sind.

2.3. Die Installation

1. Legen Sie ein Programm-Verzeichnis an.
   (Z.B. `C:\Programme\regain`).

2. Kopieren Sie folgende Dateien in dieses Verzeichnis:
   - `regain.jar` (Enthält den Crawler)
   - `log4j.properties` (Enthält die Protokollierungskonfiguration)
   - `CrawlerConfiguration.xml` (Enthält die Crawlerkonfiguration)
   - `jacob.dll` (Enthält den nativen Anteil der Jacob-API)

3. Ändern Sie die `CrawlerConfiguration.xml` nach Ihren Bedürfnissen.
   Datails siehe nächster Abschnitt.

4. Erstellen Sie das Verzeichnis, in das der Index erstellt werden soll. Das wird aus Sicherheitsgründen nicht automatisch vom Crawler erledigt.

2.4. Die Konfiguration

Der Crawler wird über zwei Dateien konfiguriert:

- Die Datei `log4j.properties`. Sie enthält alle Einstellungen, die das Protokollieren betreffen.
- Die Datei `CrawlerConfiguration.xml`. Sie enthält alle restlichen Einstellungen.

In der Datei `log4j.properties` kann man festlegen, mit welcher Granularität, mit welchem Format und wohin protokolliert werden soll.

Weitere Informationen hierzu erhalten sie unter:
http://jakarta.apache.org/log4j/docs/manual.html

Eine komplette Liste mit allen XML-Tags der Datei `CrawlerConfiguration.xml` finden Sie im Anhang A „Die XML-Tags der CrawlerConfiguration.xml“ ab Seite 19.

Hinweis:
Der Crawler arbeitet viel mit regulären Ausdrücken (kurz: Regex). Falls Sie keine Erfahrungen mit dieser Technik haben, dann lesen Sie bitte Abschnitt 2.7 „Exkurs: Reguläre Ausdrücke, Seite 10.

2.5. Der Start

Der Crawler wird von der Konsole mit dem Befehl

```
java –jar regain.jar
```

gestartet.

Dabei können folgende Parameter angegeben werden:

- `--help`: Zeigt die möglichen Aufrufparameter
- `-forceNewIndex`: Erzwingt die Erstellung eines neuen Index. Anderenfalls wird versucht, einen bereits bestehenden Index zu aktualisieren.
  Siehe Abschnitt 2.1.4 „Inkrementelle Indizierung“, Seite 6.
- `-onlyEntries <Whitelist-Eintrag1>,<Whitelist-Eintrag2>`: Die Liste der Whitelist-Einträge, die bearbeitet werden sollen. Alle anderen Einträge in

der Weißen Liste werden zwar im Index belassen, jedoch nicht aktualisiert. Siehe Abschnitt 2.1.5 „Partielle Indizierung“, Seite 6.

- `-config <Dateiname>`: Gibt die zu nutzende Konfigurationsdatei an. Default ist: `CrawlerConfiguration.xml`.
- `-logConfig <Dateiname>`: Gibt die zu nutzende Logging-Konfigurationsdatei an. Default ist: `log4j.properties`.

Beispiel mit Parametern:

```
java –jar regain.jar –config HompageConfig.xml
```

2.6. Das Indexverzeichnis

Das Indexverzeichnis kann bis zu fünf Unterverzeichnisse beinhalten, die jeweils einen Suchindex beinhalten.

Falls in der Konfiguration `<writeAnalysisFiles>` auf `true` gesetzt wurde, dann enthalten diese Verzeichnisse wiederum ein Unterverzeichnis mit dem Namen `analysis`, welches die Analysedateien enthält. Diese dienen, wie der Name schon sagt, lediglich der Analyse und können bei Nicht-Bedarf auch gelöscht werden.

Falls beim Erstellen des Index tote Links (dead Links) gefunden wurden oder Fehler auftraten, werden diese in einem Unterverzeichnis namens `log` in die Datei `deadlinks.txt` bzw. `errors.txt` geschrieben.

Jedes Unterverzeichnis hat eine genau definierte Funktion:

Verzeichnis `temp`:
Wird vom Crawler genutzt, während er einen neuen Index aufbaut. Sobald er damit fertig ist, nennt er diese Verzeichnis in `new` um.

Verzeichnis `quarantine`:
Enthält den neuen Suchindex, wenn bei der Erstellung fatale Fehler aufgetreten sind. Fatale Fehler sind, wenn der Index leer ist bzw. wenn die Anzahl der fehlerhaften Dokumente einen bestimmten Prozentsatz übersteigt.

Falls sie den Index trotzdem nutzen wollen, dann benennen Sie das Verzeichnis in `new` um.

Verzeichnis `new`:
Enthält den neuen Suchindex, bis dieser von der Suchmaske übernommen wird. Die Suchmaske prüft alle 10 Sekunden, ob dieses Verzeichnis existiert. Wenn ja, dann legt es vom gerade genutzten Index eine Sicherheitskopie an (Verzeichnis `index` wird in `backup` umbenannt) und nennt das Verzeichnis `new` in `index` um.

Verzeichnis `index`:
Enthält den Index, auf dem die Suchmaske gerade arbeitet.

Verzeichnis `backup`:
Enthält eine Sicherheitskopie des zuletzt genutzten Index. Falls Sie feststellen sollten, dass der gerade genutzte Index fehlerhaft ist, so können sie das Verzeichnis `backup` in `new` umbenennen. Es wird dann innerhalb von 10 Sekunden wieder von der Suchmaske übernommen. (Der fehlerhafte Index steht nach der Übernahme im Verzeichnis `backup`).

Hinweis:
Wie Sie den Beschreibungen entnehmen können, sind im Regelfall nur die Verzeichnisse `index` und `backup` vorhanden.

2.7. Exkurs: Reguläre Ausdrücke

Der Crawler arbeitet viel mit regulären Ausdrücken (kurz: Regex).

Falls Sie keine Erfahrung mit dieser Technik haben, finden Sie eine kleine Einführung im PDF-Dokument Regex.pdf. Hinweis: Java nutzt den gleichen Regex-Dialekt wie Perl.

Achtung:
In der XML-Konfigurationsdatei müssen XML-Zeichen, wie `&` oder `<` durch entsprechende Entitäten (`&` oder `<`) ersetzt werden!

Beispiel:
Die Regex `<a[^>]*>Der Link</a>` muss in der XML-Datei folgendermaßen angegeben werden: `&lt;a[^&gt;]*&gt;Der&amp;nbsp;Link&lt;/a&gt;`.

Regex-Gruppen:

Teile von regulären Ausdrücken können zu sog. „Gruppen“ zusammengefasst werden. Eine Gruppe wird durch eine aufgehende und eine schließende Klammer gekennzeichnet.

Jede Gruppe hat eine eindeutige Nummer, mit der sie identifiziert werden kann.

Gruppen werden nach der öffnenden Klammer nummeriert:
Die gesamte Regex hat die Nummer 0. Die erste Gruppe hat die Nummer 1. Die erste Gruppe innerhalb der Gruppe 1 hat die Nummer 2, usw.

Beispiel:

```
a(b(a(b|c)a)a(b|c)*)c      Nummer 0
 (b(a(b|c)a)a(b|c)*)       Nummer 1
   (a(b|c)a)               Nummer 2
     (b|c)                 Nummer 3
            (b|c)          Nummer 4
```

3. Die Suchmaske

Die Suchmaske wurde mit Java Server Pages (kurz: JSPs) in Verbindung mit Tag-Libraries (kurz: Taglibs) realisiert.

Sie nimmt eine Suchanfrage entgegen und zeigt die Treffer seitenweise an. Das Aussehen der Ergebnisseite ist durch die Datei `SearchOutput.jsp` weitreichend anpassbar. Es wurde darauf geachtet, dass jeder einzelne Ausgabewert von einem eigenen Taglib-Tag erzeugt wird.

3.1. Abhängigkeiten

Die Suchmaske läuft unter Java ab der Version 1.2.2.

Sie funktioniert mit Tomcat ab der Version 3.2.3.
Das impliziert folgende Versionsvorgaben:

- Taglib 1.1
- Servlet 2.2

Sie nutzt die folgenden Bibliotheken:

- Jakarta Lucene 1.4.2 (`lucene-1.4.2.jar`). Enthält den Kern der Suche, also Indexerstellung und Suche auf dem Index.
  Siehe http://jakarta.apache.org/lucene/

3.2. Die Installation

1. Installieren Sie Tomcat 3.2.3 oder höher

2. Konfigurieren Sie in der Datei `web.xml` wo sich die Konfigurationsdatei `SearchConfiguration.xml` befindet. Die Voreinstellung ist so gewählt, dass sie sich im Tomcat-Verzeichnis unter `conf/regain` befinden muss.

3. Ändern Sie die `SearchConfiguration.xml` nach Ihren Bedürfnissen. Datails siehe nächster Abschnitt.

4. Kopieren Sie die Datei `regain.war` in das Tomcat-Unterverzeichnis `webapps`.

3.3. Die Konfiguration

Die Suchmaske kann an folgenden Stellen konfiguriert werden:

- Die Datei `SearchOutput.jsp`. Sie bestimmt das Aussehen der Suchergebnisse.
- Die Datei `ErrorPage.jsp`. Sie bestimmt das Aussehen der Fehlerseite. Diese wird gezeigt, wenn beim Suchen ein Fehler auftritt.
- Die Datei `web.xml`. Sie bestimmt, wo sich die Konfigurationsdatei `SearchConfiguration.xml` befindet.
- Die Datei `SearchConfiguration.xml`. Sie enthält die eigentliche Konfiguration. Details siehe weiter unten.

Hinweis:
Die Datei `SearchInput.jsp` enthält keine TagLib-Tags. Sie zeigt nur ein Beispiel, wie man ein Suchfeld in eine ganz normale HTML-Seite einbinden kann.

Wo finde ich die Dateien?

Bis auf die `SearchConfiguration.xml` befinden sich alle Dateien in der Datei `regain.war`, welche in Wahrheit eine ZIP-Datei ist. `SearchOutput.jsp`, `SearchInput.jsp` und `ErrorPage.jsp` befinden sich direkt im Hauptverzeichnis, web.xml in einem Unterverzeichnis namens `WEB-INF`.

Um sie zu bearbeiten kann man folgendermaßen vorgehen:

- `regain.war` nach `regain.war.zip` umbenennen.
- Mit einem ZIP-Programm entpacken.
- Dateien ändern.
- Mit einem ZIP-Programm wieder verpacken. Dabei ist darauf zu achten, dass die Verzeichnisstruktur erhalten bleibt. Insbesondere darf innerhalb der ZIP-Datei kein neues Oberverzeichnis erzeugt werden.
- Wieder nach `regain.war` umbenennen.

Die `SearchConfiguration.xml` enthält die eigentliche Konfiguration und ist, da sie sich nicht in der `regain.war` befindet leicht zugänglich. TODO

Eine komplette Liste mit allen XML-Tags der Datei `SearchConfiguration.xml` finden Sie im Anhang C „Die XML-Tags der SearchConfiguration.xml“ ab Seite 30.

3.4. Der Start

Starten Sie den Tomcat. Rufen Sie dazu im Tomcat-Unterverzeichnis `bin` die Datei `startup.bat` auf. Falls es per Doppelklick nicht funktioniert, dann versuchen Sie es von der Konsole.

Tippen Sie dazu:
```
cd %TOMCAT_HOME%\bin
startup
```

Rufen Sie nun die Suchseite auf:

http://localhost:8080/regain/SearchInput.jsp
oder
http://localhost:8080/regain/SearchOutput.jsp

Hinweis:
Wenn eine JSP zum ersten mal geladen wird, dann braucht dies einige Zeit. Das ist normal und kommt daher, dass Tomcat die JSP erst übersetzen muss.

3.5. Die Syntax für Suchanfragen von Lucene

Die Anfragen an die Suchmaske werden von Lucene nach einer recht mächtigen Syntax geparst. Lucene bietet zwar die Möglichkeit, auch eine eigene Syntax zu verwenden, die Standardsyntax sollte jedoch im Normalfall mehr als ausreichen.

An dieser Stelle wird nur auf die wichtigsten Elemente dieser Syntax eingegangen.

Eine vollständige Beschreibung finden Sie hier:
http://jakarta.apache.org/lucene/docs/queryparsersyntax.html

Terme

Eine Suchanfrage besteht aus Termen und Operatoren. Es gibt zwei Arten von Termen: Einfache Terme und Phrasen.

Ein einfacher Term ist ein einzelnes Wort, wie `Test` oder `Hallo`.

Eine Phrase ist eine Gruppe von Worten, die in Hochkommas eingeschlossen sind, beispielsweise `“Hallo Otto“`. Sie treffen auf Worte zu, die in genau dieser Reihenfolge vorkommen.

Operatoren

Terme werden mit Hilfe von Operatoren verbunden. Die wichtigsten werden nun vorgestellt.

Operator	**OR**
Beschreibung	OR verbindet zwei Terme und findet Dokumente, die wenigstens einen der Terme beinhalten.
Hinweis	OR ist der Standardoperator, d.h. er wird genutzt, wenn kein Operator angegeben wird.

Beispiel	Um Dokumente zu suchen, die Otto Maier oder Handbuch beinhalten:
	`“Otto Maier“ OR Handbuch`
	oder einfach:
	`“Otto Maier“ Handbuch`

Operator	**AND**
Beschreibung	AND trifft auf Dokumente zu, die beide Terme irgendwo im Text beinhalten.
Beispiel	Um Dokumente zu suchen, die sowohl Otto Maier als auch Handbuch beinhalten: `“Otto Maier“ AND Handbuch`

Operator	**+**
Beschreibung	+ gibt an, dass der folgende Term erforderlich ist.
Beispiel	Liefert Ergebnisse, die Tester enthalten müssen und Duft enthalten dürfen: `Duft +Tester`

Operator	**-**
Beschreibung	- schließt Dokumente aus, die den folgenden Term beinhalten.
Beispiel	Sucht Dokumente, die Foto enthalten, Suche jedoch nicht: `Foto -Suche`

Wildcards

Man kann Wildcards (auch bekannt als Jokerzeichen) für einzelne oder für viele Buchstaben setzen.

Um ein einzelnes Zeichen offen zu lassen, verwendet man das Fragezeichen (?).

Um viele Zeichen offen zu lassen, nutzt man den Stern (*).

Um `Text` oder `Test` zu suchen:

```
Te?t
```

Um `Test`, `Tester` oder `Testament` zu suchen:

```
Test*
```

Ein Wildcard kann auch in der Mitte eines Terms stehen:

```
Te*t
```

Hinweis: Ein Term darf nicht mit einem `?` oder einem `*` beginnen.

Unscharfe Suche

Um Worte zu finden, die ähnlich buchstabiert werden, kann man an einen Term eine Tilde (~) anhängen.

Der folgende Ausdruck findet auch `Mayer`, `Meyer` oder `Maier`:

```
Meier~
```

4. Was ist bei anderen Sprachen zu beachten

Lucene kann prinzipiell für jede Sprache genutzt werden. Da die Texte in Unicode verarbeitet werden, sind auch Sprachen mit nicht lateinischen Buchstaben wie Russisch, Chinesisch oder Japanisch möglich.

Die Mischung mehrerer Sprachen in einem Index ist nicht möglich und wäre im Übrigen auch nicht nützlich.

Um eine andere Sprache zu nutzen, muss man nur einen anderen Analyzer nutzen. Zur Erinnerung: Der Analyzer versucht, ein Wort auf seinen Wortstamm zurückzuführen. Auf diese Weise wird z.B. bei der Suche von `Katze` auch `Kätzchen` gefunden.

Dabei gibt es eine wichtige Grundregel: Da im Index aus Effizienzgründen nur noch die Wortstämme – wie z.B. `katz` – liegen, muss man bei der Indizierung den selben Analyzer verwenden, wie bei der Suche. Dies wird gewährleistet, indem der verwendete Analyzer im Index in die Datei `analyzerType.txt` geschrieben wird. Die Suchmaske liest diese Datei aus und kann so feststellen, welcher Analyzer zur Erzeugung des Index verwendet wurde.

Welcher Analyzer genutzt werden soll, lässt sich in der Crawler-Konfiguration angeben (Tag `analyzerType`).

Momentan sind folgende Analyzer verwendbar:

- `english`: Für englische Dokumente
- `german`: Für deutsche Dokumente

Man kann weitere Anlayzer-Typen hinzufügen, indem man die Methode `createAnalyzer(...)` der Klasse `LuceneToolkit` erweitert. Diese Methode wird sowohl vom Crawler als auch von der Suchmaske genutzt.

Wichtig: Nach einer Änderung des Analyzers muss einen eventuell schon vorhandenen Index komplett neu erstellen.

5. Bewertung von Lucene

Lucene zeichnet sich durch einige Vorteile aus: Es ist leicht an allen Ecken und Enden erweiterbar und anpassbar, sehr flexibel einzusetzen und geht sparsam mit Systemressourcen um. Letzteres betrifft sowohl Bearbeitungszeit als auch Speicherplatzverbrauch.

Außerdem verfügt Lucene über eine mächtige Syntax für Suchanfragen. Siehe Abschnitt 3.5 „Die Syntax für Suchanfragen von Lucene“, Seite 14.

Lucene stellt allerdings nur den Kern einer Suche, also die Erstellung und die Nutzung eines Index zur Verfügung. Die Darstellung der Suchergebnisse und die Aufbereitung der zu indizierenden Inhalte muss man selbst übernehmen. Gerade der letzte Punkt ist angesichts vieler verschiedener Dokumentenformate nicht gerade trivial.

Anhang A: Die XML-Tags der CrawlerConfiguration.xml

Übersicht:

- <configuration>
  - <proxy>
    - <host>
    - <port>
    - <user>
    - <password>
  - <loadUnparsedUrls>
  - <httpTimeout>
  - <useLinkTextAsTitleList>
    - <urlPattern> *
  - <searchIndex>
    - <dir>
    - <buildIndex>
    - <writeAnalysisFiles>
    - <maxFailedDocuments>
    - <stopwordList>
    - <exclusionList>
  - <controlFiles>
    - <finishedWithoutFatalsFile>
    - <finishedWithFatalsFile>
  - <preparatorList>
    - <preparator> *
      - <urlPattern>
      - <class>
      - <config>
        - <section name="..."> *
          - <param name="..."> *
  - <htmlParserPatternList>
    - <pattern parse="..." index="..." regexGroup="..."> *
  - <directoryParserPatternList>
    - <pattern index="..."> *
  - <startlist>
    - <start parse="..." index="..."> *
  - <blacklist>
    - <prefix> *
  - <whitelist>
    - <prefix> *
  - <auxiliaryFieldList>
    - <auxiliaryField name="..." regexGroup="..."> *

Die Datei `CrawlerConfiguration.xml` bietet die folgenden XML-Tags:

Tag	`<proxy>`
Aufgabe	Stellt ein, welcher Proxy-Server genutzt werden soll.
Werte	Kennt die folgenden Kind-Tags: • `<host>`: Der Host-Name des Proxy • `<port>`: Der Port des Proxy • `<user>`: Der Benutzername, falls eine Anmeldung nötig ist. • `<password>`: Das Passwort, falls eine Anmeldung nötig ist. Alle diese Felder können auch leer gelassen werden, falls kein Proxy oder keine Anmeldung nötig sind. Falls Sie nicht wissen, was sie einstellen sollen, dann übernehmen Sie die Einstellungen Ihrem Browser.

Tag	`<host>` (Kind-Tag von `<proxy>`)
Aufgabe	Der Host-Name des Proxy. Kann komplett weggelassen werden, wenn kein Proxy nötig ist.
Werte	Eine beliebige Zeichenkette
Beispiel	`proxy.something.de` oder `10.10.10.53`

Tag	`<port>` (Kind-Tag von `<proxy>`)
Aufgabe	Der Port des Proxy. Kann komplett weggelassen werden, wenn kein Proxy nötig ist.
Werte	Eine Zahl
Beispiel	`8080`

Tag	`<user>` (Kind-Tag von `<proxy>`)
Aufgabe	Der Benutzername für die Anmeldung beim Proxy. Kann komplett weggelassen werden, wenn keine Anmeldung nötig ist.
Werte	Eine beliebige Zeichenkette
Beispiel	`karl34`

Tag	`<password>` (Kind-Tag von `<proxy>`)
Aufgabe	Das Passwort für die Anmeldung beim Proxy. Kann komplett weggelassen werden, wenn keine Anmeldung nötig ist.
Werte	Eine beliebige Zeichenkette
Beispiel	`i56dFg2s`

Tag	`<loadUnparsedUrls>`
Aufgabe	Gibt an, ob URLs geladen werden sollen, die weder durchsucht noch indiziert werden. Auf diese Weise kann geprüft werden, ob diese URLs Tote Links sind. Diese Funktion kann auch dazu verwendet werden, den Zwischenspeichern (Cache) des Servers zu füllen.
Werte	`true` oder `false`
Beispiel	`false`

Tag	`<httpTimeout>`
Aufgabe	Der Timeout für HTTP-Downloads. Dieser Wert bestimmt die maximale Zeit in Sekunden, die ein HTTP-Download insgesamt dauern darf.
Werte	Ganzzahl
Beispiel	`180`

Tag	`<useLinkTextAsTitleList>`
Aufgabe	Enthält die Liste der regulären Ausdrücke, auf die die URL eines Dokuments passen muss, damit anstatt des wirklichen Dokumententitels der Text des Links, der auf das Dokument gezeigt hat, als Dokumententitel genutzt wird.
Werte	Beliebig viele `<urlPattern>`–Kind-Tags.

Tag	`<urlPattern>` (Kind-Tag von `<useLinkTextAsTitleList>`)
Aufgabe	Ein Regulärer Ausdruck, der ein Dokument bestimmt, für das der Linktext als Titel genommen werden soll.
Werte	Regulärer Ausdruck
Beispiel	`^http://.*\.(pdf|xls|doc|rtf)$`

Tag	`<searchIndex>`
Aufgabe	Enthält alle Einstellungen, die den Suchindex-Verzeichnis betreffen.
Werte	Kennt die folgenden Kind-Tags: • `<dir>`: Das Verzeichnis, in dem der Suchindex stehen soll. • `<buildIndex>`: Gibt an, ob der Suchindex überhaupt erstellt werden soll. • `<analyzerType>`: Der zu verwendende Analyzer. • `<writeAnalysisFiles>`: Gibt an, ob Analyse-Deteien geschrieben werden sollen. • `<maxFailedDocuments>`: Gibt den maximalen Prozentsatz von gescheiterten Dokumenten an. • `<stopwordList>`: Eine Liste von Worten, die nicht indiziert werden sollen. • `<excludeList>`: Eine Liste von Worten die bei der Indizierung nicht vom Analyzer verändert werden sollen.

Tag	`<dir>` (Kind-Tag von `<searchIndex>`)
Aufgabe	Das Verzeichnis, in dem der Suchindex am Ende stehen soll.
Werte	Ein Verzeichnisname
Beispiel	`C:\regain\index`

Tag	`<buildIndex>` (Kind-Tag von `<searchIndex>`)
Aufgabe	Gibt an, ob der Suchindex überhaupt erstellt werden soll.
Werte	`true` oder `false`
Beispiel	`true`

Tag	`<analyzerType>` (Kind-Tag von `<searchIndex>`)
Aufgabe	Der zu verwendende Analyzer. Details siehe Abschnitt 4 „Was ist bei anderen Sprachen zu beachten“, Seite 17.
Werte	`standard` oder `german`
Beispiel	`german`

Tag	`<writeAnalysisFiles>` (Kind-Tag von `<searchIndex>`)
Aufgabe	Gibt an, ob Analyse-Deteien geschrieben werden sollen. Diese Dateien helfen, die Qualität der Index-Erstellung zu prüfen und werden in einem Unterverzeichnis im Index-Verzeichnis angelegt.
Werte	`true` oder `false`
Beispiel	`false`

Tag	`<maxFailedDocuments>` (Kind-Tag von `<searchIndex>`)
Aufgabe	Gibt den maximalen Prozentsatz von gescheiterten Dokumenten an. Ist das Verhältnis von gescheiterten Dokumenten zur Gesamtzahl von Dokumenten größer als dieser Prozentsatz, so wird der Index verworfen. Gescheiterte Dokumente sind Dokumente die es entweder nicht gibt (Deadlink) oder die nicht ausgelesen werden konnten.
Werte	Gleitkommazahl zwischen 0 und 100
Beispiel	`9.5`

Tag	`<stopwordList>` (Kind-Tag von `<searchIndex>`)
Aufgabe	Eine Liste von Worten, die nicht indiziert werden sollen. Eine solche Liste hilft, den Index klein zu halten. Nehmen Sie hier häufige Worte auf, deren Indizierung keinen Sinn ergeben würde.
Werte	Wortliste, durch Leerzeichen getrennt
Beispiel	`wir und oder ohne mit am im in aus`

Tag	`<excludeList>` (Kind-Tag von `<searchIndex>`)
Aufgabe	Eine Liste von Worten die bei der Indizierung nicht vom Analyzer verändert werden sollen. Der Analyzer versucht, für jedes Wort eine Grundform zu finden, so dass z.B. `schöne` und `schönes` die gleichen Treffer liefert. Der Analyzer wird sowohl bei der Indexerstellung als auch bei einer Suchanfrage genutzt. Es mag Worte geben, bei denen das Probleme bereitet. Nehmen Sie diese in dieser Liste auf.
Werte	Wortliste, durch Leerzeichen getrennt
Beispiel	`SehrProblematischesWort NochProblematischeresWort`

Tag	`<controlFiles>`
Aufgabe	Enthält die Namen der Kontrolldateien.
Werte	Dieser Tag ist optional. Kennt die folgenden Kind-Tags: • `<finishedWithoutFatalsFile>`: Der Name der Kontrolldatei für erfolgreiche Indexerstellung. • `<finishedWithFatalsFile>`: Der Name der Kontrolldatei für fehlerhafte Indexerstellung.

Tag	<finishedWithoutFatalsFile> (Kind-Tag von <controlFiles>)
Aufgabe	Dieser Tag ist optional. Der Name der Kontrolldatei für erfolgreiche Indexerstellung. Diese Datei wird erzeugt, wenn der Index erstellt wurde, ohne dass fatale Fehler aufgetreten sind.
Werte	Ein Dateiname
Beispiel	`C:\lucene\control\NoFatals`

Tag	<finishedWithFatalsFile> (Kind-Tag von <controlFiles>)
Aufgabe	Dieser Tag ist optional. Der Name der Kontrolldatei für fehlerhafte Indexerstellung. Diese Datei wird erzeugt, wenn der Index erstellt wurde, wobei fatale Fehler aufgetreten sind.
Werte	Ein Dateiname
Beispiel	`C:\lucene\control\WithFatals`

Tag	<preparatorList>
Aufgabe	Enthält die Liste der Präparatoren.
Werte	Beliebig viele <preparator>–Kind-Tags.

Tag	<preparator> (Kind-Tag von <preparatorList>)
Aufgabe	Enthält die Einstellungen für einen Präparator.
Werte	Kennt die folgenden Kind-Tags: • <urlPattern>: Der Reguläre Ausdruck, zu dem die URL eines Dokuments passen muss, damit es von diesem Präparator bearbeitet wird. • <class>: Der Klassenname des Präparators. • <config>: Der Konfiguration des Präparators.

Tag	<urlPattern> (Kind-Tag von <preparator>)
Aufgabe	Der Reguläre Ausdruck, zu dem die URL eines Dokuments passen muss, damit es von diesem Präparator bearbeitet wird.
Werte	Regulärer Ausdruck
Beispiel	`\.(/|html|htm)$`

Tag	<class> (Kind-Tag von <preparator>)
Aufgabe	Der Klassenname des Präparators. Die Klasse muss die Schnittstelle `de.filiadata.lucene.spider.document.Preparator` implementieren und über einen Standardkonstruktor (also ein Konstruktor ohne Parameter) verfügen.
Werte	Klassenname
Beispiel	`de.filiadata.lucene.spider.document.HtmlPreparator`

Tag	<config> (Kind-Tag von <preparator>)
Aufgabe	Die Konfiguration des Präparators.
Werte	Kennt die folgenden Kind-Tags: • <section>: Ein Abschnitt der Präparator-Konfiguration.

Tag	<section> (Kind-Tag von <config>)
Aufgabe	Ein Abschnitt der Präparator-Konfiguration. Welche Abschnitte unterstützt werden, hängt vom jeweiligen Präparator ab.
Werte	Kennt die folgenden Kind-Tags: • <param>: Ein Parameter der Präparator-Konfiguration.
Attribut `name`	Der Name des Abschnitts. (Werte: Abschnittsname)

Tag	<param> (Kind-Tag von <section>)
Aufgabe	in Parameter der Präparator-Konfiguration. Welche Parameter unterstützt werden, hängt vom jeweiligen Präparator ab.
Werte	Der konfigurierte Wert.
Attribut `name`	Der Name des Abschnitts. (Werte: Abschnittsname)

Tag	<htmlParserPatternList>
Aufgabe	Liste mit Suchmustern, die beim Durchsuchen eines HTML-Dokuments dazu genutzt werden, um URLs zu identifizieren.
Werte	Beliebig viele <pattern>–Kind-Tags.

Tag	<pattern> (Kind-Tag von <htmlParserPatternList>)
Aufgabe	Ein Suchmuster, das einen URL-Typ finden kann.
Werte	Regulärer Ausdruck
Beispiel	`="([^"]*\.(doc|pdf|rtf))`
Attribut `parse`	Gibt an, ob ein so gefundenes Dokument selbst wieder nach URLs durchsucht werden soll. (Werte: `true` oder `false`)
Attribut `index`	Gibt an, ob ein so gefundenes Dokument in den Suchindex aufgenommen werden soll. (Werte: `true` oder `false`)

Attribut `regexGroup`	Gibt die Nummer derjenigen Regex-Gruppe an, die die URL beinhaltet. (Werte: Eine Zahl)

Tag	`<directoryParserPatternList>`
Aufgabe	Liste mit Suchmustern, die beim Durchsuchen eines Verzeichnisses dazu genutzt werden, um zu entscheiden, ob und wie eine Datei vom Crawler bearbeitet wird.
Werte	Beliebig viele `<pattern>`–Kind-Tags.

Tag	`<pattern>` (Kind-Tag von `<directoryParserPatternList>`)
Aufgabe	Ein Suchmuster, das entscheidet, wie eine Datei vom Crawler bearbeitet wird.
Werte	Regulärer Ausdruck
Beispiel	`="([^"]*\.(doc|pdf|rtf))`
Attribut `index`	Gibt an, ob eine zum Regulären Ausdruck passende Datei in den Suchindex aufgenommen werden soll. (Werte: `true` oder `false`)

Tag	`<startlist>`
Aufgabe	Liste mit URLs, bei denen der Crawler-Prozeß starten soll.
Werte	Beliebig viele `<start>`–Kind-Tags.

Tag	`<start>` (Kind-Tag von `<startlist>`)
Aufgabe	Start-URL, bei der Crawler-Przeß beginnt.
Werte	URL
Beispiel	`http://www.dm-drogeriemarkt.de/CDA/Home/` oder `file://P:/dokumente/`
Attribut `parse`	Gibt an, ob das Dokument nach URLs durchsucht werden soll. (Werte: `true` oder `false`)
Attribut `index`	Gibt an, ob das Dokument in den Suchindex aufgenommen werden soll. (Werte: `true` oder `false`)

Tag	`<blacklist>`
Aufgabe	Die Schwarze Liste. Sie enthält Präfixe, die eine URL *nicht* haben darf, um bearbeitet zu werden.
Werte	Beliebig viele `<prefix>`–Kind-Tags.

Tag	`<whitelist>`
Aufgabe	Die Weiße Liste. Sie enthält Präfixe, von denen eine URL einen haben *muß*, um bearbeitet zu werden.
Werte	Beliebig viele `<prefix>`–Kind-Tags.

Tag	`<prefix>` (Kind-Tag von `<blacklist>` oder `<whitelist>`)
Aufgabe	Gibt einen URL-Präfix an.
Werte	URL-Präfix
Beispiel	`http://www.dm-drogeriemarkt.de/CDA/Home/Suchen/`

Tag	`<auxiliaryFieldList>`

Aufgabe	Die Liste der Zusatzfelder.
	
	Der Index kann durch Zusatzfelder erweitert werden, die aus der URL eines Dokuments generiert werden.
	
	Beispiel: Angenommen Sie haben ein Verzeichnis mit einem Unterverzeichnis für jedes Projekt. Dann könnten Sie daraus ein Feld mit dem Projektnamen generieren.
	
	Das folgende Tag generiert das Zusatzfeld `project` mit dem Wert `otto23` aus der URL `file://c:/projects/otto23/docs/Spez.doc`:
	
	`<auxiliaryField name="`**`project`**`" regexGroup="`**`1`**`">`
	`  `**`^file://c:/projects/([^/]*)`**
	`</auxiliaryField>`
	
	URLs, die nicht zum Regulären Ausdruck passen, bekommen kein `project`-Zusatzfeld.
	
	Dadurch bekommen Sie bei der Suche nach `Angebot project:otto23` nur Treffer aus diesem Verzeichnis.

Werte	Kennt die folgenden Kind-Tags: • `<auxiliaryField>`: Die Definition für ein Zusatzfeld.

Tag	`<auxiliaryField>` (Kind-Tag von `<auxiliaryFieldList>`)
Aufgabe	Die Definition für ein Zusatzfeld.
Werte	Regulärer Ausdruck
Beispiel	`^file://c:/projects/([^/]*)`
Attribut `name`	Gibt den Namen des Zusatzfelds an. (Beispiel: `projct` oder `system`)
Attribut `regexGroup`	Gibt die Nummer derjenigen Regex-Gruppe an, die den Wert beinhaltet, den das Zusatzfeld bekommen soll. (Werte: Eine Zahl)

Anhang B: Die Konfiguration der Präparatoren

Um einen Präparator zu konfigurieren, muss man in der `CrawlerConfiguration.xml` unterhalb des entsprechenden `<preparator>`-Tags einen `<config>`-Tag hinzufügen. Siehe Anhang A „Die XML-Tags der CrawlerConfiguration.xml“ ab Seite 19.

Eine Präperator-Konfiguration besteht aus Abschnitten (sections), die jeweils Parameter enthalten.

Beispiel: Die folgende Konfiguration enthält die Abschnitte sec1 und sec2. Der Abschnitt sec1 hat die Parametern p1 und p2, der Abschnitt sec2 enthält den Parameter p3:

```
<config>
  <section name="sec1">
    <param name="p1">a value</param>
    <param name="p2">another value</param>
  </section>
  <section name="sec2">
    <param name="p3">42</param>
  </section>
</config>
```

Alle Präparatoren, die hier nicht aufgeführt sind, sind nicht konfigurierbar.

B.1. Konfiguration des HtmlPreparator

Abschnitt `contentExtractor`

Enthält die Einstellungen für einen Content-Extrahierer. Es lassen sich mehrere Extraktoren für verschiedene URL-Präfixe erstellen, um für verschiedene Seiten-Typen verschiedene Extraktoren nutzen zu können.

Die Parameter:

- Parameter `prefix`: Der URL-Präfix. Eine URL muss mit diesem Präfix beginnen, damit dieser Extraktor auf sie angewendet wird.
  Beispiel: `http://www.murfman.de/`
- Parameter `startRegex`: Der Reguläre Ausdruck, der die Stelle findet, wo in einem HTML-Dokument der zu indizierende Inhalt beginnt. Wenn Sie diesen Parameter weglassen, dann wird der gesamte Anfang des HTML-Dokuments indiziert.
  Beispiel: `&lt;td class="content"`
- Parameter `endRegex`: Der Reguläre Ausdruck, der die Stelle findet, wo in einem HTML-Dokument der zu indizierende Inhalt endet. Wenn Sie diesen Parameter

weglassen, dann wird das gesamte Ende des HTML-Dokuments indiziert.
Beispiel: `</table>`

- Parameter `headlineRegex`: Der reguläre Ausdruck, der eine Überschrift identifiziert. Die so gewonnenen Überschriften werden im Feld `headlines` indiziert. Auf diese Weise werden Überschriften bei der Suche stärker gewichtet. Wenn Sie diesen Parameter weglassen, dann wird das HTML-Dokument nicht nach Überschriften durchsucht.
- Parameter `headlineRegex.group`: Gibt die Nummer derjenigen Regex-Gruppe an, die die Überschrift enthält.
  Beispiel: `1`

Abschnitt `pathExtractor`

Enthält die Einstellungen für einen Pfad-Extrahierer. Es lassen sich mehrere Extraktoren für verschiedene URL-Präfixe erstellen, um für verschiedene Seiten-Typen verschiedene Extraktoren nutzen zu können.

Die Parameter:

- Parameter `prefix`: Der URL-Präfix. Eine URL muss mit diesem Präfix beginnen, damit dieser Extraktor auf sie angewendet wird.
  Beispiel: `http://www.murfman.de/`
- Parameter `startRegex`: Der Reguläre Ausdruck, der die Stelle findet, wo in einem HTML-Dokument der Navigationspfad beginnt.
  Beispiel: `<!-- Start Navigationspfad -->`
- Parameter `endRegex`: Der Reguläre Ausdruck, der die Stelle findet, wo in einem HTML-Dokument der Navigationspfad endet.
  Beispiel: `<!-- Ende Navigationspfad -->`
- Parameter `pathNodeRegex`: Der reguläre Ausdruck, der einen Teil des Pfades identifiziert. Dieser Ausdruck wird nur auf den Teil des HTML-Dokuments angewendet, der zwischen der `startRegex` und der `endRegex`. Damit werden falsche Treffer vermieden.
  Beispiel: `<a.*href="([^"]*)">(.*)</a>`
- Parameter `pathNodeRegex.urlGroup`: Gibt die Nummer derjenigen Regex-Gruppe an, die die URL beinhaltet.
  Beispiel: `1`
- Parameter `pathNodeRegex.titleGroup`: Gibt die Nummer derjenigen Regex-Gruppe an, die den Titel beinhaltet.
  Beispiel: `2`

Anhang C: Die XML-Tags der SearchConfiguration.xml

Übersicht:

- <configuration>
  - <indexList>
    - <defaultSettings>
      - ... (Kaskadierte Einstellungen)
    - <index name="..."> *
      - <dir>
      - <openInNewWindowRegex>
      - <searchFieldList>
      - <rewriteRules>
        - <rule prefix="..." replacement="..."> *

Kaskadierte Einstellungen

In der `SearchConfiguration.xml` werden die Einstellungen vorgenommen, die für die Suchmaske wichtig sind.

Die Suchmaske unterstützt das Suchen auf mehreren Indizes. Zu jedem Index kann man in der Konfiguration angeben, wo er liegt und wie Treffer aus diesem Index angezeigt werden sollen.

Alle Einstellungen, bis auf die Lage des Index, können kaskadiert erfolgen. D.h. wenn eine Einstellung nicht im jeweiligen `<index>`-Tag gemacht wurde, dann wird geprüft, ob sie im `<defaultSettings>`-Tag gemacht wurde. Wurde auch hier nichts eingestellt, dann wird der Defaultwert genutzt.

Auf diese Weise lassen sich Standardeinstellungen festlegen, die für alle Indizes gelten. Will man für einen bestimmten Index doch eine andere Einstellung, dann kann man die Standardeinstellung überschreiben, indem man im entsprechenden `<index>`-Tag etwas anderes angibt.

Die Lage der `SearchConfiguration.xml`

Damit die Suchmaske die `SearchConfiguration.xml` finden kann, muss in der web.xml angegeben werden, wo sie sich befindet. Die Standardeinstellung ist so gewählt, dass sich die `SearchConfiguration.xml` im Tomcat-Verzeichnis unter `conf/regain` befinden muss.

Die Datei `SearchConfiguration.xml` bietet die folgenden XML-Tags:

Tag	`<indexList>`
Aufgabe	Enthält die Einstellungen zu den Indizes
Werte	Kennt die folgenden Kind-Tags: • `<defaultSettings>`: Die kaskadierten Standardeinstellungen. • `<index>`: Die Einstellungen zu einem Index.

Tag	`<defaultSettings>` (Kind-Tag von `<indexList>`)
Aufgabe	Die kaskadierten Standardeinstellungen.
Werte	Alle Tags, die auch im `<index>`-Tag erlaubt sind, außer dem `<dir>`-Tag.

Tag	`<index>` (Kind-Tag von `<indexList>`)
Aufgabe	Die Einstellungen zu einem Index.
Werte	Kennt die folgenden Kind-Tags: • `<dir>`: Das Verzeichnis, in dem der Index liegt. • `<openInNewWindowRegex>`: Der Reguläre Ausdruck, der URLs identifiziert, die in einem neuen Browserfenster geöffnet werden sollen. • `<searchFieldList>`: Die Indexfelder, in denen standardmäßig gesucht werden soll. • `<rewriteRules>`: Die Regeln für das URL-Rewriting.
Attribut `name`	Der Name des Index. (Werte: Indexname)

Tag	`<dir>` (Kind-Tag von `<index>` oder `<defaultSettings>`)
Aufgabe	Das Verzeichnis, in dem der Index liegt.
Werte	Verzeichnis
Beispiel	`C:/regain/searchindex_big`

Tag	`<openInNewWindowRegex>` (Kind-Tag von `<index>` oder `<defaultSettings>`)
Aufgabe	Der Reguläre Ausdruck, der URLs identifiziert, die in einem neuen Browserfenster geöffnet werden sollen.
Werte	Regulärer Ausdruck
Beispiel	`.(pdf|rtf|doc|xls|ppt)$`

Tag	`<searchFieldList>` (Kind-Tag von `<index>` oder `<defaultSettings>`)
Aufgabe	Die Indexfelder, in denen standardmäßig gesucht werden soll. Hinweis: Der Benutzer kann außerdem mit Hilfe des `field-`Operators auch in anderen Feldern suchen. Siehe Lucene Query-Syntax: http://jakarta.apache.org/lucene/docs/queryparsersyntax.html
Werte	Liste von Indexfeldnamen
Beispiel	`content title headlines`

Tag	`<rewriteRules>` (Kind-Tag von `<index>` oder `<defaultSettings>`)
Aufgabe	Die Regeln für das URL-Rewriting. Für die Anzeige der Treffer kann sog. URL-Rewriting eingesetzt werden. D.h. dass die URL des Dokuments verändert wird. So kann man z.B. Dokumente in `file://c:/www-data/intranet/docs` indizieren und im Browser als `http://intranet.murfman.de/docs` anzeigen lassen. Trifft auf eine URL keine der Regeln zu, dann bleibt sie unverändert.
Werte	Kennt die folgenden Kind-Tags: • `<rule>`: Eine Ersetzungsregel.

Tag	`<rule>` (Kind-Tag von `<rewriteRules>`)
Aufgabe	Eine Ersetzungsregel für das URL-Rewriting. Beispiel: Mit den hier gezeigten Beispielwerten wird die URL `file://c:/www-data/intranet/docs/manual/regain.pdf` in `http://intranet.murfman.de/docs/manual/regain.pdf` geändert.
Attribut `prefix`	Der URL-Präfix, der ersetzt werden soll. (Werte: URL-Präfix, z.B.: `file://c:/www-data/intranet/docs`)
Attribut `replacement`	Der URL-Präfix, der statt dessen gezeigt werden soll. (Werte: URL-Präfix, z.B.: `http://intranet.murfman.de/docs`)

Index

Analysedateien 5, 9
Analyzer 17
analyzerType.txt 17
Andere Sprachen 17
Crawler 4
Crawler-Prozeß 4
CrawlerConfiguration.xml 7, 20
  <analyzerType> 22
  <blacklist> 26
  <buildIndex> 22
  <class> 25
  <config> 25
  <controlFiles> 23
  <dir> 22
  <directoryParserPatternList> 26
  <excludeList> 23
  <finishedWithFatalsFile> 24
  <finishedWithoutFatalsFile> 24
  <host> 20
  <htmlParserPatternList> 25
  <httpTimeout> 21
  <loadUnparsedUrls> 21
  <maxFailedDocuments> 23
  <param> 25
  <password> 20
  <pattern> 25, 26
  <port> 20
  <prefix> 26
  <preparator> 24
  <preparatorList> 24
  <proxy> 20
  <searchIndex> 22
  <section> 25
  <start> 26
  <startlist> 26
  <stopwordList> 23
  <urlPattern> 21, 24
  <useLinkTextAsTitleList> 21
  <user> 20
  <whitelist> 26
  <writeAnalysisFiles> 22
CrawlerJob 4
dead Links 9
ErrorPage.jsp 13
Fehlerliste 4

Gruppe 10
B.1.HtmlPreparator 27
contentExtractor 27
endRegex 27, 28
headlineRegex 28
headlineRegex.group 28
pathExtractor 28
pathNodeRegex 28
pathNodeRegex.titleGroup 28
pathNodeRegex.urlGroup 28
prefix 27, 28
startRegex 27, 28
Index 9
Index aktualisieren 6
indexDir 13
Indexierung 5
Inkrementelle Indizierung 6
jacob.dll 7
jacob.jar 7
Jacobgen 7
jakarta-regexp-1.3.jar 7
log4j-1.2.9.jar 7
log4j.properties 7
Lucene 5, 18
lucene-1.4.2.jar 7
Mehrsprachigkeit 17
openInNewWindowRegex 13
Partielle Indexierung 6
PDFBox-0.6.7a.jar 7
poi-2.5.1-final-20040804.jar 7
poi-scratchpad-2.5.1-final-20040804.jar 7
Präperator 5
Querysyntax 15
regain.jar 7
regain.war 13
Regex 10
Regex-Gruppe 10
Regulärer Ausdruck 10
Schwarze Liste 5
SearchConfiguration.xml 30
<defaultSettings> 30
<dir> 30
<index> 30
<indexList> 30
<openInNewWindowRegex> 30
<rewriteRules> 31
<rule> 31
<searchFieldList> 31

searchFieldList 13
SearchOutput.jsp 13
Suchindex 9
Suchmaske 12
Syntax für Suchanfragen 15
web.xml 13
Weiße Liste 5
xercesImpl.jar 7
xml-apis.jar 7