**LUCENE IN ACTION**

Mục lục:

[Phần 1 : INDEXING 3](#_Toc234419209)

[1. Quy trình đánh chỉ mục 3](#_Toc234419210)

[2. Các toán tử đánh chỉ mục cơ bản 5](#_Toc234419211)

[3. Khuếch đại các tài liệu và các trường(Boosting Documents and Fields) 5](#_Toc234419212)

[4. Điều khiển tiến trình đánh chỉ mục 6](#_Toc234419213)

[5. Bộ nhớ trong trong quá trình đánh chỉ mục : RAMDirectory 8](#_Toc234419214)

[6. Tối ưu hóa việc đánh chỉ mục 8](#_Toc234419215)

[7. Tính đồng thời,an toàn tiến tình,ngăn chặn các thực thi 9](#_Toc234419216)

[Phần 2 : SEARCH:CHỨC NĂNG TÌM KIẾM TRONG ỨNG DỤNG 11](#_Toc234419217)

[1. Bộ chuyển đổi câu truy vấn của người dùng : QueryParser 12](#_Toc234419218)

[Các biểu thức truy vấn của QueryParser 14](#_Toc234419219)

[2. Các toán tử luận lí 15](#_Toc234419220)

[3. Cụm truy vấn(phrase queries) 16](#_Toc234419221)

[4. Sử dụng lớp IndexSearcher 16](#_Toc234419222)

[5. Tìm hiểu về các loại Query 18](#_Toc234419223)

[Phần 3 : ANALYSIS 22](#_Toc234419224)

[1. Using analyzers 23](#_Toc234419225)

[2. Analyzing the analyzer 26](#_Toc234419226)

[3. Using the built-in analyzers 32](#_Toc234419227)

[4. Dealing with keyword fields 33](#_Toc234419228)

[5. “Sounds like” querying 33](#_Toc234419229)

[6. Synonyms, aliases, and words that mean the same 35](#_Toc234419230)

[7. Stemming analysis 35](#_Toc234419231)

[8. Language analysis issues 36](#_Toc234419232)

[Phần 4 : ADVANCED SEARCH TECHNIQUES 37](#_Toc234419233)

[1. Indexing a Microsoft Word document 37](#_Toc234419234)

[2. Indexing an RTF document 38](#_Toc234419235)

[3. Indexing a plain-text document 38](#_Toc234419236)

[4. Creating a document – handling framwork 39](#_Toc234419237)

[5. Other text-extraction tools 41](#_Toc234419238)

[Document-management systems and services 42](#_Toc234419239)

[Phần 5 : EXTENFING SEARCH 43](#_Toc234419240)

[Using a custom sort method 43](#_Toc234419241)

[Phần 6 : PARSING COMMON DOCUMENT FORMAT 44](#_Toc234419242)

[1. Handling rich-text document 44](#_Toc234419243)

[2. Indexing XML 45](#_Toc234419244)

[3. Indexing một DPF document 46](#_Toc234419245)

[4. Indexing an HTML document 46](#_Toc234419246)

[5. Indexing a Microsoft Word document 46](#_Toc234419247)

[6. Indexing an RTF document 47](#_Toc234419248)

[7. Indexing a plain-text document 48](#_Toc234419249)

[8. Creating a document – handling framwork 48](#_Toc234419250)

[9. Other text-extraction tools 50](#_Toc234419251)

[Document-management systems and services 51](#_Toc234419252)

[Phần 7 : TOOLS AND EXTENSIONS 52](#_Toc234419253)

[1. Playing in Lucene’s Sandbox 52](#_Toc234419254)

[2. Interacting with an index 53](#_Toc234419255)

[3. Analyzers, tokenizers, and TokenFilters,oh my 55](#_Toc234419256)

[4. Java Development with Ant and Lucene 56](#_Toc234419257)

[5. JavaScript browser utilities 57](#_Toc234419258)

[6. Synonyms from WordNet 59](#_Toc234419259)

[7. Highlighting query terms 61](#_Toc234419260)

[8. Chaining filters 62](#_Toc234419261)

[9. Sorting an index in Berkeley DB 62](#_Toc234419262)

[Phần 8 : LUCENE PORTS 63](#_Toc234419263)

[1. Ports’ relation to Lucene 63](#_Toc234419264)

[2. Clucene 64](#_Toc234419265)

[3. DotLucene 64](#_Toc234419266)

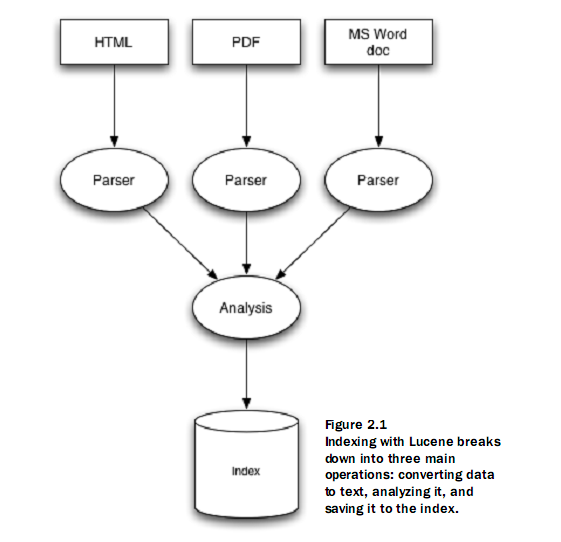
[4. Plucene 65](#_Toc234419267)

[5. Lupy 66](#_Toc234419268)

[6. PyLucene 67](#_Toc234419269)

# INDEXING

### Quy trình đánh chỉ mục



#### Convert to text

Để tiến hành index được trong Lucene,thì trước hết ta phải chuyển đổi dữ liệu thành dạng văn bản thuần túy(plain text,như file .txt chẳng hạn).Điều này là quan trọng,bởi vì dữ liệu được lưu trữ dưới nhiều dạng file khác nhau (pdf,word,excel,powerpoint,html…),trong khi đó Lucene chỉ hỗ trợ bạn index các Field dạng String , Date hoặc đối tượng Reader mà thôi.Vấn đề chuyển đổi thành dạng plain text sẽ được nói rõ hơn trong phần Chương 7

#### Analysis

Mỗi khi bạn chuẩn bị cho việc index và tạo ra đối tượng Document với các Field,thì Lucene sẽ phân tích dữ liệu này sao cho phù hợp nhất với việc index.Để làm điều này,Lucene sẽ phân chia dữ liệu thành các chuỗi hoặc là các kí tự thông qua việc lựa chọn các toán tử thực thi trên chúng.Chẳng hạn như việc bạn phân tích thành các kí tự thường,hoặc bỏ đi các từ ngữ không có nghĩa…Chi tiết về việc phân tích cùng các lớp API sẽ được đề cập đến trong Chương 4

#### Wrting index

Sau khi dữ liệu được phân tích ,nó sẽ sẵn sang cho việc index.Lucene sẽ chứa dữ liệu này theo cấu trúc inverted index(chỉ mục có thể nghịch đảo ).Cấu trúc này sẽ có hiệu quả để tiết kiệm dung lượng ổ đĩa và cho phép tìm kiếm nhanh hơn các từ khóa trong quá trình search.Nguyên tắc của nó là thay vì phải tìm kiếm các từ nào chứa trong tài liệu đó thì với cấu trúc này sẽ tối ưu hóa việc tìm ra câu trả lời “tài liệu nào chứa từ khóa này”.

### Các toán tử đánh chỉ mục cơ bản

Lucene hỗ trợ các toán tử giúp thực hiện việc đánh chỉ mục như:

* Thêm tài liệu mới (Document) cùng các trường (Fields): Keyworks,UnIndexed,UnStored và Text.Trong mỗi tài liệu lại có thể có chứa nhiều Fields cùng tồn tại,và trong mỗi Fields lại có thêm nhiều giá trị khác nhau
* Xóa tài liệu ra khỏi chỉ mục : (Remove Documents )Sử dụng lớp IndexReader với phương thức delete () ta có thể dễ dàng xóa bỏ tài liệu được chỉ định ra khỏi chỉ mục.Lucene sẽ xem như các tài liệu này được đánh dấu như là sẽ xóa.Tuy nhiên việc này chỉ có thể thực hiện khi gọi hàm close().
* Hủy bỏ việc xóa tài liệu (Undeleting Documents): Như đã đề cập,việc thực sự xóa bỏ tài liệu khi gọi phương thức close() của lớp IndexReader.Do đó trươc khi thực sự xóa tài liệu này thì chỉ cần gọi phương thức undeleteAll() của cùng lớp IndexReader trước đó sẽ xóa bỏ đi các tài liệu được đánh dấu xóa bỏ.
* Cập nhật tài liệu : Lucene không hỗ trợ ta thưc hiện việc cập nhật tài liệu ,thay vào đó ta sẽ xóa bỏ tài liệu và sau đó thêm lại tài liệu mới thay thế.Chú ý rằng để đảm bảo tốc độ thực thi thì tốt nhất việc xóa bỏ và thêm tài liệu mới nên thực hiện theo khối,không nên xen lẫn giữa việc xóa và thêm tài liệu mới.

### Khuếch đại các tài liệu và các trường(Boosting Documents and Fields)

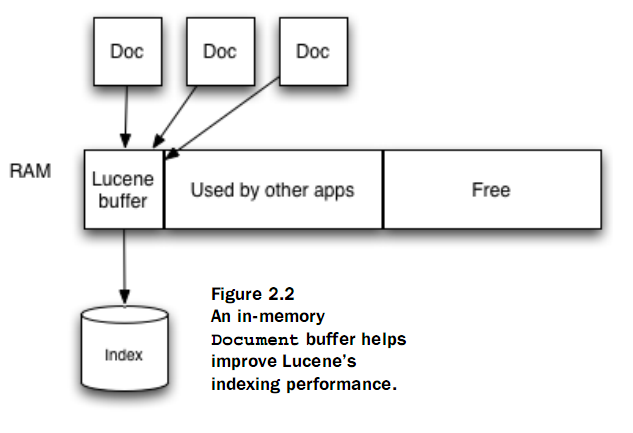
Trong quá trình index không phải tất cả các Tài liệu và trường (Documents and Fields) cũng có độ ưu tiên như nhau,goi là điểm ưu tiên (Score).Nếu như ta có ý định sắp xếp kết quả trả về trong quá trình tìm kiếm theo nhưng tiêu chí nào đó chẳng hạn như là sắp xếp theo điểm ưu tiên thì chúng ta cần phải boost tài liệu hoặc trường cần sắp xếp.

Mặc định tất cả các tại liệu đều không có boost hoặc là boost là 1.0.Do đó muốn thay đổi độ ưu tiên này ta chỉ cần gọi hàm setBoost(float) cho Document hoặc Field nào đó mà ta mong muốn.Độ ưu tiên (độ boost) càng cao thì tài liệu càng quan trọng và ngược lại .

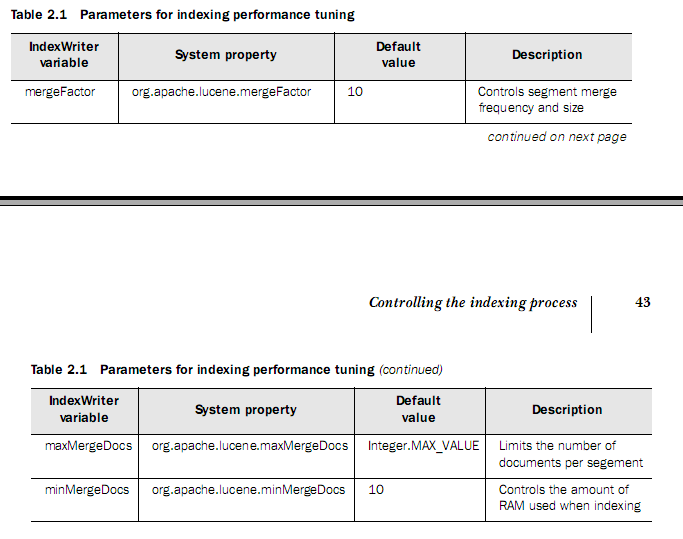
### Điều khiển tiến trình đánh chỉ mục

Với các ứng dụng nhỏ thì việc index dữ liệu là không lớn,ta hoàn toàn có thể mặc định index theo cài đặt của Lucene.Tuy nhiên với các ứng dụng lơn,khi mà dung lượng tại liệu lên đến hàng triệu,thì tốc độ index là điều hết sức quan trọng,thay vì phải mất hàng giờ thì có thể chỉ mất hàng phút mà thôi.

* Điều chỉnh việc thực thi đánh chỉ mục : Trong các ứng dụng đánh chỉ mục điển hình thì trờ ngại chính là việc ghi chỉ mục xuống đĩa.Vì vậy ta phải điều chỉnh làm sao cho Lucene tự “thông minh” trong việc đánh chỉ mục với các tài liệu mới cũng như bổ sung các file có sẵn.



Như ta đã thấy,mỗi khi thêm tài liệu mới vào chỉ mục,chúng sẽ được khởi tạo vào trong bộ nhớ đệm thay vì ghi trực tiếp xuống đĩa.Sử dụng bộ nhớ đệm này có nhiều lí do và quan trong hàng đầu vẫn là cải thiện tốc độ index.Lucene cho phép bạn điều chỉnh các tham số này một cách dễ dàng.



* Các tham số :
  + mergeFactor: tham số cho phép ta điều chỉnh số lượng Document sẽ chứa trong bộ nhớ trước khi ghi vào đĩa cũng như là bao lâu thì sẽ trộn những segment này với nhau ,giá trị mặc đinh là 10.Điều này có nghĩa là số segment trên đĩa sẽ là mũ của 10.

Ví dụ giá trị này là 10 thì có nghĩa là sẽ tạo ra 10 segment,mỗi segment có thể chứa 10 Documents,và khi segment thứ 10 được tao ra thì chúng sẽ được merge thành 1 segment có kích thước là 100. Và cứ như thế.

* + maxMergeDocs :Đây là tham số giới hạn số lượng Document tối đa chứa trong segment trước khi merger các segments với nhau.
  + minMergeDocs : tương tự như maxMergeDocs,song nó chỉ ra có tối thiểu bao nhiêu tài liệu chứa trong segment trước khi merge các segment lại với nhau

Như vậy,ta thấy rằng: nếu tăng giá trị mergeFactor thì Lucene sẽ sử dụng nhiều bộ nhớ chính hơn nhưng mật độ ghi xuống đĩa lại ít thường xuyên hơn,Điều này sẽ tăng tốc độ tiến trình index .Tuy nhiên nó sẽ tạo ra nhiều file index hơn và lại làm ảnh hưởng tới việc tìm kiếm về sau,khi phải truy xuất nhiều file trong thư mục index.maxMergeDocs càng cao thì thích hợp cho các dữ liệu khối hơn.

Tóm lại mergeFactor và maxMergeDocs càng có giá trị lớn thì tốc độ index sẽ cao,song đừng quá lạm dụng vì có thể ảnh hưởng tới việc tìm kiếm.

### Bộ nhớ trong trong quá trình đánh chỉ mục : RAMDirectory

Như đã đề cập,Lucene lưu Document mới vừa thêm vào trong bộ đệm buffer trước khi ghi xuống đĩa.Mặc định Lucene sẽ sử dụng lớp FSDirectory- một lớp cài đặt của Directory.tuy nhiên nếu ta muốn kiểm soát tốt hơn việc sử dụng bộ nhớ và mật độ ghi xuống đĩa thì RAMDirectory là một thay thế.

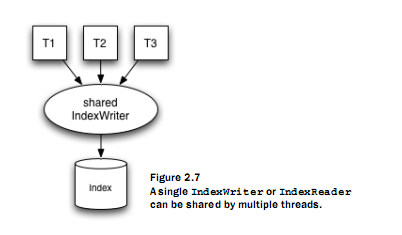
FSDirectory làm việc trực tiếp trên đĩa cứng,trong khi đó RAMDirectory thực thi trong bộ nhớ chính vì thế mà tốc độ sẽ nhanh hơn.Từ đây ta có thể nghĩ ngay tới việc sử dụng kết hợp giữa hai lớp trên để cải thiện tốc độ index.sử dụng RAMDirectory như bộ đệm lưu giữ tài liệu mới thêm vào tới một số lượng nhất định,sau đó đẩy xuống cho lớp FSDirectory thưc thi ghi xuống đĩa.

### Tối ưu hóa việc đánh chỉ mục

Việc tối ưu hóa tiến trình đánh chỉ mục là tiến trình trộn nhiều file chỉ mục lại với nhau để giảm thiểu thời gian đọc chỉ mục tr ong quá trình tìm kiếm.Bằng việc sử dụng API của lucene mà cụ thể là hàm optimize() của đối tượng IndexWriter ta có thể dễ dàng tối ưu điêu này.Tuy nhiên việc làm này chỉ có hiệu quả tăng tốc độ tìm kiếm trên chỉ mục đã có,mà không có tác động tới tốc độ đánh chỉ mục.

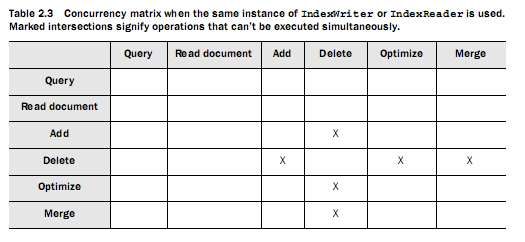
### Tính đồng thời,an toàn tiến tình,ngăn chặn các thực thi

* Các luật đồng thời : Lucene cung cấp cho ta nhiều toán tử liên quan tới việc đánh chỉ mục tài liệu như : xóa,cập nhật.Do đó trong quá trình thực hiện chúng ta phải tuân theo một số luật cụ thể để tránh việc đụng độ trong quá trình thực thi.Điều này là cấn thiết khi mà có nhiều thực thi diễn ra một cách thường xuyên trước những yêu cầu gởi từ web tới ứng dụng của bạn.Sau đây là một số luật cơ bản :
  + Bất kì toán tử chỉ đọc nào cũng có thể thực thi đồng thời,Chẳng hạn là nhiều tiến trình có thể tìm kiếm cùng một chỉ mục tại một thời điểm
  + Bất kì toán tử chỉ đọc nào cũng có thể thực thi đồng thời trong khi một chỉ mục nào đó đang được cập nhật.Ví dụ: người dùng có thể tìm kiếm trong chỉ mục trong khi nó đang được cập nhật,thêm tài liệu mới hoặc là được xóa khỏi chỉ mục
  + Chỉ có duy nhất 1 toán tử cập nhật chỉ mục có thể thực thi tại một thời điểm.Một chỉ mục chỉ có thể được mở bởi chỉ một đối tượng IndexWriter hoặc là IndexReader tại một thời điểm mà thôi.
* Tính an toàn của tiến trình(thread-safety) :Mặc dù ta thấy rằng việc cập nhật chỉ mục tại một thời điểm sử dụng nhiều đối tượng IndexWriter hoặc là IndexReader là không được phép.Song,cùng một đối tượng có thể được chia sẽ bởi nhiều tiến trình khác nhau và tất cả cùng gọi tới những phương thức cập nhật chỉ mục sẽ được đồng bộ hóa sao cho việc thực thi có tính tuần tự.



Lucene luôn đảm bảo rằng các toán tử cập nhật không được chồng lắp nhau.Có nghĩa là trước khi thêm tài liệu mới vào chỉ mục,ta phải đóng tất cả các thể hiện IndexReader trên cùng chỉ mục.Tương tự,trước khi xóa hoặc cập nhật tài liệu trong chỉ mục,ta phải đóng tất cả thể hiện IndexWriter đang mở trên cùng chỉ mục.

Sau đây là bảng mô tả các toán tử có thể thực thi đồng thời hay không:



* Khóa chỉ mục: nhằm tránh gây ra sự đụng độ trong quá trình sử dụng các hàm API của Lucene,Thư viện này đã tạo ra các file lock bên cạnh các segment để đảm bảo rằng việc thực thi một chương trình tại một thời điểm.Mỗi chỉ mục có một tập các file lock.mặc nhiên tất cả các file này được tạo ra trong thư mục tạm được xác định bởi thuộc tính java.io.tmpdir.

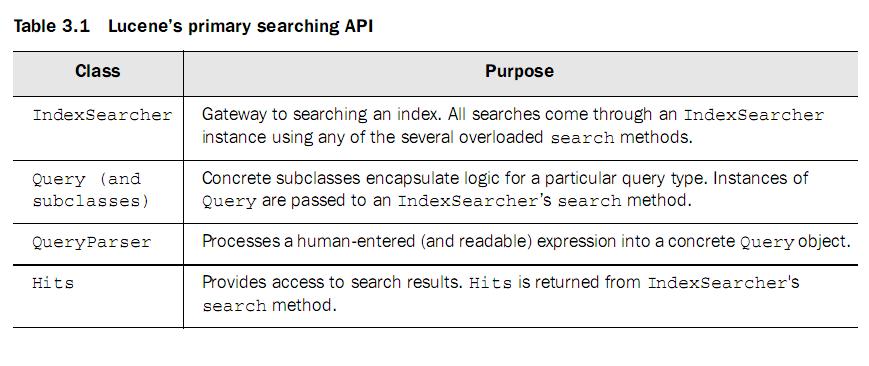
Có hai loại file lock được tao ra:

* + File Write.lock : được tạo ra trong quá trình đánh chỉ mục tài liệu.nó nhằm đảm bảo các tiến trình thực thi một cách có tuần tự trong quá trình cập nhật chỉ mục.Hơn nữa nó được tồn tại khi dối tượng IndexWriter đang duy trì và giữ cho tới khi nó đóng mới thôi. Ngoài ra nó tồn tại khi đối tượng IndexReader đang được sử dụng để xóa,hủy bỏ việc xóa,hay cài đặt các mục của trường nào đó.nó còn giúp khóa chỉ mục mà diễn ra lâu hơn mong đợi.
  + File commit.lock : tạo ra trong quá trình merge các segment.Nó được dùng bất kể khi nào những segment đang được đọc hoặc trộn lại với nhau.Nó được nắm giữ bởi đối tượng IndexReader trước khi nó nó đọc các segment và chỉ giải phóng sau khi IndexReader đã mở và đọc các segment
* Vô hiệu hóa tính năng khóa chỉ mục : nhiều khi ta cần vô hiệu hóa tính năng khóa chỉ mục của Lucene.chẳng hạn là ứng dụng cần đọc chỉ mục nằm trên ổ CD-ROM.Có nghĩa là ứng dụng chỉ ở chế độ đọc mà thôi,tức là chỉ tìm kiếm trên đó mà không hề có cập nhật chỉ mục.Để vô hiệu hóa ta chỉ cấn thay đổi thuộc tính disableLuceneLocks thành true là được.

# SEARCH:CHỨC NĂNG TÌM KIẾM TRONG ỨNG DỤNG

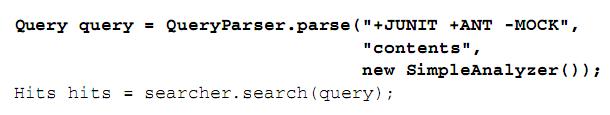
Chương này sẽ giới thiệu về việc thêm chức năng search vào ứng dụng một khi đã có dữ liệu đã được đánh chỉ mục(index).Chúng tôi sẽ trình bày một số lớp cơ bản trong Lucene hỗ trợ bạn việc tìm kiếm mà nhóm đã ứng dụng,với những chức năng cao cấp hơn sẽ được thảo luận ở các chương sau.

Bảng sau mô tả các lớp chính dùng để search trong Lucene API :



### Bộ chuyển đổi câu truy vấn của người dùng : QueryParser

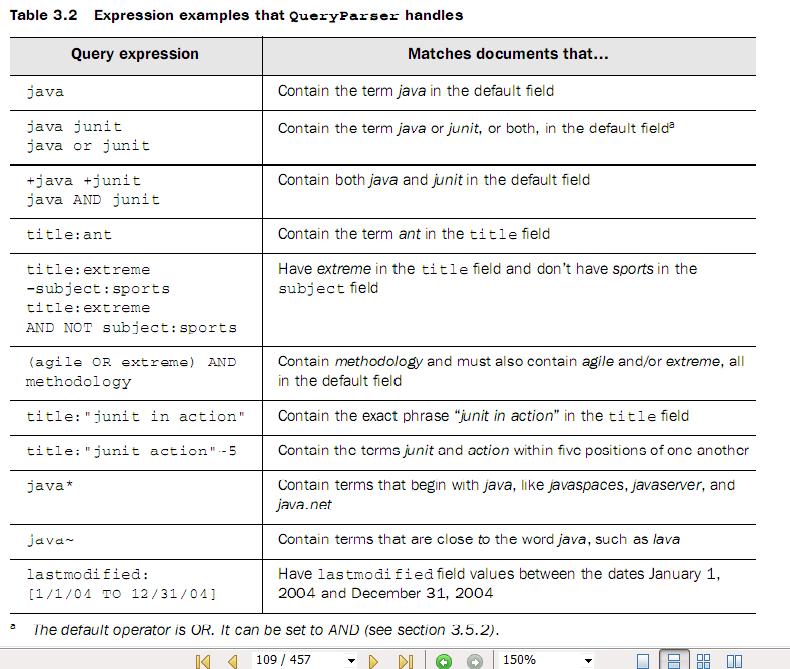
Hai yêu cầu quan trọng trong ứng dụng tìm kiếm đòi hỏi là : chuyển đổi câu truy vấn và truy xuất thông tin trả về. Hầu hết các phương thức Lucene đòi hỏi đối tượng Query.Việc chuyển đổi câu truy vấn là việc biểu diễn câu truy vấn của người dùng thành đối tượng Query phù hợp để sau đó truyền vào hàm tìm kiếm của lucene.Lucene có thể tìm ra kết quả chỉ khi câu truy vấn truyền vào là đúng định dạng của nó. Chẳng hạn như ví dụ sau:



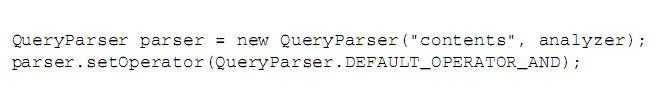
Để thực hiện được việc chuyển đổi câu truy vấn của người dùng,QueryParser cần thêm một đối tượng khác gọi là bộ phân tích Analyzer-sẽ đề cập trong chương sau.Ở đây ta chỉ cần biết là tùy vào việc chọn lựa bộ Analyzer để phân tích chuỗi truyền vào thì kết quả sẽ khác nhau.Điều ta cần nhớ là index dùng bộ Analyzer nào thì khi tìm kiếm ta nên dùng lại bộ Analyzer đó.

Như ví dụ trên,người dùng truy vấn bằng chữ viết hoa,song trong dữ liệu đánh chỉ mục đã lưu trữ thì lại là chữ thường.Do đó dùng SimpleAnalyzer –đã sử dụng trong việc index, sẽ chuyển đổi câu truy vấn thành chữ thường và vì thế sẽ tìm ra kết quả trong quá trình tìm kiếm.

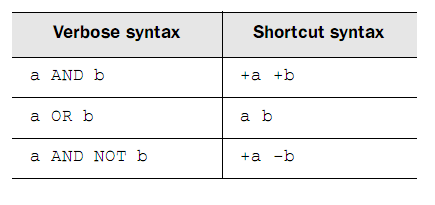
### Các biểu thức truy vấn của QueryParser



### Các toán tử luận lí

QueryParser sử dụng nhiều toán tử luận lí để thực hiện việc chuyển đổi câu truy vấn như : OR ,AND,NOT.Mặc định là OR.Chẳng hạn câu truy vấn sau : abc xyz thì sẽ được phân tích thành là abc or xyz or (abc and xyz).Để thay đổi tham số mặc định này,ta cần đặt lại toán tử cho đối tượng QueryParser.

Ngoài ra ta có thể sử dụng các từ viết tắt thay thế cho các toán tử này ví dụ như bảng dưới đây:



Tất nhiên là ta hoàn toàn có thể gom nhóm câu truy vấn lại ví dụ như là (a and b) or c…bằng ngoặc đơn.Khi đó sẽ hiểu là tìm tài liệu có chứa cả a và b hoặc là chỉ c mà thôi

Việc phân tích câu truy vấn nhằm mục đích tìm kiếm ra tài liệu có chứa từ cần tìm.Tuy nhiên việc đánh chỉ mục diễn ra trên nhiều trường khác nhau (Fields).Do đó muốn tìm tài liệu ứng với trường nào đã đánh chỉ mục thì bạn cần xác định rõ ràng.Như ví dụ trên là tìm trong trường “contents” của chỉ mục.

### Cụm truy vấn(phrase queries)

Những câu truy vấn được bao đóng trong dấu nháy kép sẽ tạo ra một cụm truy vấn.Các từ trong dấu nháy sẽ được phân tích,vì thế kết quả có thể sẽ không chính xác như cụm từ ban đầu.(chưa hoàn thành)

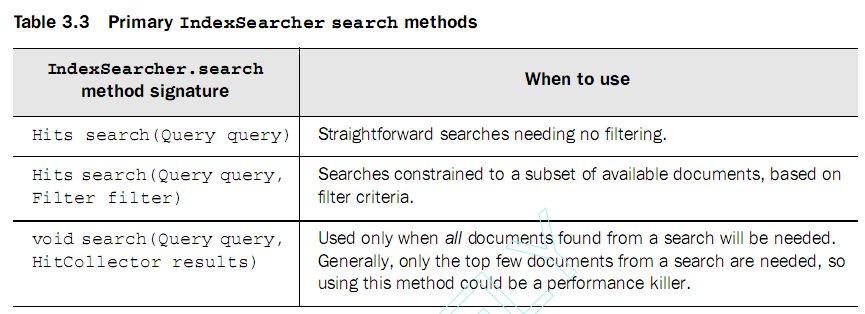
### Sử dụng lớp IndexSearcher

Tiếp theo chúng ta sẽ xem xét tới lớp IndexSearcher của Lecene.Giống như hầu hết các hàm API khác,nó rất dễ sử dụng.Việc khởi tạo thể hiện IndexSearcher thực hiện bằng các tham số khác nhau

* Bằng Directory chứa dữ liệu đã index
* Bằng đường dẫn tới file hệ thống



Sau khi tạo ra đối tượng IndexSearcher,ta sẽ gọi phương thức search để thực hiện việc tìm kiếm.Có ba phương thức chính để tìm kiếm.Song ta chủ yếu sử dụng phương thức search(Query),tức tham số là câu truy vấn Query

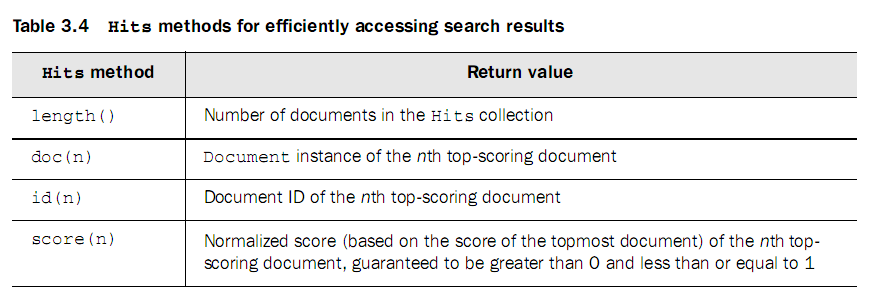


Như ta đã thấy,các phương thức tìm kiếm này đều trả về là các Hits –chứa các thông tin đã tìm kiếm được,Sau đây ta sẽ tìm hiểu thêm về Hits

#### Tìm hiểu về Hits

Kết quả trả về là đối tượng Hits và được sắp xếp theo thứ tự độ chính xác.Thông qua đối tượng này ta có thể truy xuất thêm nhiều thông tin về kết quả tìm kiếm.

Sau đây sẽ điểm qua bốn phương thức chính của đối tượng này,được liệt kê dưới bảng đây.



Mặc định 100 Documents sẽ tự động được khởi tạo ban đầu và sẽ được xử lí.bộ Hits sẽ tự nó thêm vào khi người dùng truy vấn tới những tài liệu ở mức trên.Tuy nhiên điển hình vẫn là việc xem xét các tại liệu có độ ưu tiên cao hơn mà thôi.

#### Phân trang thông qua Hits

Phân trang kết quả tìm kiếm là điều hết sức cần thiết trong việc trình bày kết quả trả về.Có hai hướng cài đặt chính:

* Giữ đối tượng Hits và IndexSearcher trong khi người dùng chuyển kết quả tìm kiếm
* Thực hiện truy vấn lại mỗi khi người dùng chuyển đến trang mới

Truy vấn lại thường được dùng hơn và là giải pháp tối ưu hơn.Việc này đòi hỏi phải lưu trữ trạng thái người dùng.Trong ứng dụng web,nơi người dùng gõ truy vấn ta cần lưu lại chuỗi truy vấn ban đầu.ta có thể lưu giữ trong các hidden field hoặc là cookie.và sau mỗi lần truy vấn lại thì phỉa cập nhật lại câu truy vấn của người dùng.

Một điểm cần lưu ý là mặc dù việc tìm kiếm diễn ra trên thư mục chứa dữ liệu index,song để tăng tốc độ tìm kiếm,ta nên nạp dữ liệu đọc được từ index và đẩy lên RAMDirectory.phần này đã đề cập tới trong chương trước.

### Tìm hiểu về các loại Query

Như ta đã thấy,phương thức search của đối tượng IndexSearcher đòi hỏi tham số Query.Tùy vào ứng dụng ,mục đích của câu truy vấn là tìm kiếm gì mà ta sẽ cung cấp dạng Query tương ứng hoặc là kết hợp tất cả. Trong hấu hết các ứng dụng ta sử dụng đối tượng QueryParser để chuyển đổi câu truy vấn theo từng loại thích hợp.Lucene cung cấp bốn loại Query :QueryParse,BooleanQuery,RangeQuery và TermQuery.Sau đây.ta sẽ tìm hiểu từng loại Query và lúc nào QueryParse sẽ chuyển đổi câu truy vấn thành dạng nào.

#### Tìm kiếm theo giới hạn : TermQuery

Cách tìm kiếm chính trong chỉ mục là tìm trong giới hạn xác đinh nào đó gọi là Term.Term là một phần nhỏ trong chỉ mục,bao gồm tên trường(Field name) và giá trị kèm theo chẳng hạn như sau:



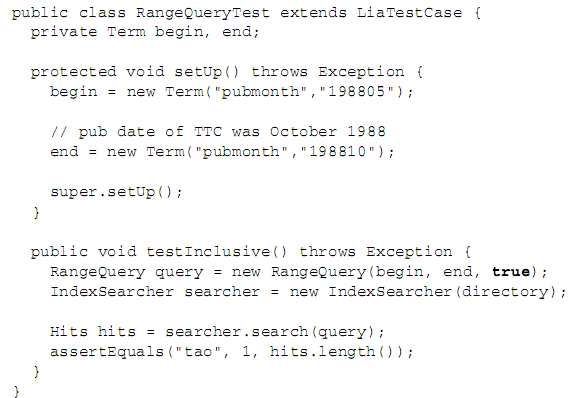
Như vậy là tất cả các kết quả nào có chứa từ khóa “java” trong phần “contents” sẽ được trả về.Chú ý rằng kiểu chữ ở đây là viết thường sẽ trùng hợp với bộ phân tích đã dùng để index dữ liệu nếu không kết quả sẽ không chính xác

Nói thêm là việc sử dụng TermQuerys rất hữu dụng khi bạn cần tìm kiếm tài liệu theo từ khóa,nếu tài liệu đã được index theo dạng là Field.Keyword().

* Nếu câu truy vấn chỉ chứa một từ thì QueryParser sẽ tạo ra TermQuery để truy vấn

#### Tìm kiếm theo dãy :RangeQuery

Trong trường hợp mà dữ liệu được index đã được sắp xếp theo thứ tự giống kiểu từ điển,thì việc tìm kiếm theo dãy tỏ ra hiệu quả hơn cả.RangeQuery của Lucene cho phép ta tìm kiếm những mục nằm trong giới hạn đầu và cuối của dãy cần tìm.

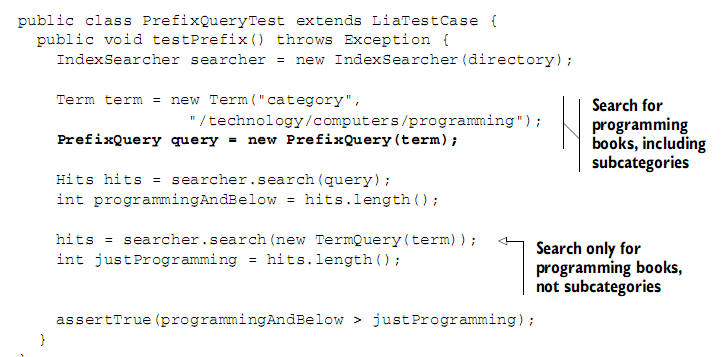


* Nếu biểu thức truy vấn có dạng :
  + [begin to end] : lấy các tài liệu nằm trong dãy này
  + {begin to end} : lấy các tài liệu nằm ngoài khoảng này

Thì QueryParse sẽ khởi tạo RangeQuery .

#### Tìm kiếm dựa trên chuỗi :PrefixQuery

Việc tìm kiếm theo kiểu này sẽ cho những kết quả chứa các tài liệu mà bắt đầu bằng chuỗi xác định.cách này thi thoảng phát huy tác dụng trong ít trường hợp chẳng hạn như là bạn có chỉ mục về danh mục sách và ta muốn tìm kiếm những cuốn sách thuộc danh mục nào đó.như ví dụ sau:



* Nếu câu truy vấn kết thúc là dấu “ \* ” thì term dạng PrefixQuery sẽ đươc tạo ra

#### Kết hợp các truy vấn : BooleanQuery

Những loại truy vấn trên có thể được kết hợp lại với nha bằng cách sư dụng đối tượng BooleanQuery với phương thức add.Các truy vấn này có thể kết hợp bằng NOT,AND hoặc là OR.

Nếu required=prohibited=true: kết hợp các truy vấn bằng toán tử AND

Nếu required=prohibited=false: kết hợp các truy vấn bằng toán tử OR

Nếu required=true và prohibited=false: kết hợp các truy vấn bằng toán tử NOT

* Nếu câu truy vấn là kết hợp của nhiều mục bao đóng bởi dấu ngoặc đơn truy vấn dạng BooleanQuery sẽ đươc tạo ra

#### Tìm kiếm theo cụm: PhraseQuery

Index chứa các thông tin về vị trí giữa các từ với nhau.PhraseQuery sử dụng các thông tin này để xác định các tài liệu chứa các từ có quan hệ với các từ khác trong cùng tài liệu.chẳng hạn, bạn đang tìm chuỗi sau:”the quick brown fox can jumped over the lazy dog”.Nếu không biết chính xác cụm truy vấn này ,ta sẽ tìm các tài liệu mà có chứa các từ “quick” và “fox” gần nhau.nếu chỉ sử dụng TermQuery-truy vấn theo từ đơn thì ta không thể tìm được tài liệu nào có hai từ kể trên gần nhau

Ta gọi khoảng cách tối đa giữa hai từ là slop và Distance là số vị trí cần di chuyển của từ đó trong cụm từ cần tìm so với cụm từ trong tài liệu.Khi đó PhraseQuery sẽ so sánh giữa các slop này để cho ra kết quả gần giống nhất

#### Tìm kiếm theo wildcard: WildcardQuery

Đây là cách tìm kiếm cho những loại truy vấn mà thiếu đi một vài từ trong đó,chẳng hạn như là tìm từ khóa “?oo\*” (đúng ra là book).

Dấu “?” có nghĩa là không có từ nào hoặc là một từ bất kì

Dấu “\*” có nghĩa là không có từ nào hoặc là nhiều từ bất kì

#### Tìm kiếm theo những từ giống nhau: FuzzyQuery

Loại truy vấn cuối cùng được đề cập tới là FuzzyQuery.Cách truy vấn này sẽ xem những từ tương tự nhau là một từ.Lucene sử dụng thuật toán về khoảng cách từ gọi là Levenshtein distance ,chi tiết có thể tham khảo tại đây (<http://www.merriampark.com/ld.htm>).

Ta chỉ cần hiểu rằng cách tính khoảng cách ở đây khác hơn so với khoảng cách trong loại truy vấn PhraseQuery,đó là số từ cần thêm.xóa để biến một từ thành từ khác.Do đó các từ tương tự nhau ở đây được xét là các từ mà xóa bỏ hày thêm một số từ nào đó để trở thành từ khác tuy nhiên mức độ ưu tiên cho các từ có số từ cần thêm vào hay bớt đi là ít hơn.

* Nếu câu truy vấn kết thúc là dấu “~” thì truy vấn dạng FuzzyQuery sẽ đươc tạo ra.

# ANALYSIS

Trong chương này những vấn đề được đặt ra và giải quyết:

* Tìm hiểu quá trình phân tích của Lucene.
* Khảo sát vấn đề quan trọng đó là QueryParse.
* Khảo sát quá trình dùng các hàm của Lucene để tạo bộ phân tích theo mục đích riêng.
* Vấn đề tích hợp các ngôn ngữ trong Lucene.

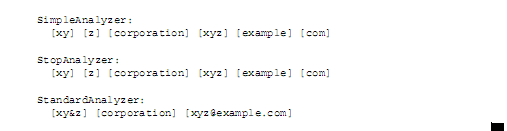
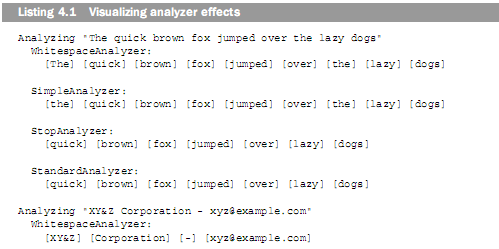
Trong Lucene , phân tích (*analysis*) là quá trình chuyển đổi các field văn bản về dạng trình bày chỉ mục cơ bản nhất (*term*).Các terms thì được sử dụng để xác định rõ tài liệu nào sẽ phù hợp với một câu truy vấn trong quá trình tìm kiếm.Bộ phân tích (*analyzer*) là cách nói tóm lược quá trình phân tích . Analyzer phân tích trong đoạn văn bản thành *tokenizes* , đó là quá trình rút trích các từ, bỏ đi hệ thống các dấu chấm câu , chuyển toàn bộ các chữ trong văn bản về dạng chữ thường ( *lowercasing* hay còn gọi là *normalizing*), loại bỏ các từ chung (*common words*) , giảm số lượng từ từ văn bản đưa vào (*root form* hay còn gọi *stremming*). Quá trình này còn được gọi là *tokenization,*chuyển đoạn văn bản thành nhiều khúc văn bản được gọi là các *token*, (sẽ được nói rõ hơn trong các phần sau) *.*Tokens được kết hợp với các field name của chúng được gọi là *terms* .

Mục đích chính của Lucene là kết quả có thể thu hồi được thông tin dễ dàng , vì vậy điều quan trọng đó là quá trình làm tinh dữ liệu từ dữ liệu thô đưa vào.Sau quá trình tạo ra terms , terms sẽ là những khối dữ liệu được dùng để tìm kiếm trực tiếp. Vì vậy chọn bộ phân tích đúng đắn là cốt yếu quan trọng của quá trình phát triển phần mềm tìm kiếm. Ngôn ngữ là một yếu tố phải được nghĩ đến để chọn bộ phân tích, bởi vì đều có đặc trưng riêng và duy nhất của từng ngôn ngữ. Kế tiếp đó là phạm vi của đoạn text , ví dụ tìm kiếm văn bản sẽ có phạm vi tìm kiếm khác với tìm kiếm media. Phải quan tâm đến các yếu tố khác như các thuật ngữ, thành ngữ, các từ viết tắt từ các bộ chữ ghép lại (ví dụ như NATO:North Atlantic Treaty Organization), … Lucene là nền tảng để chúng ta có thể tạo ra bộ tìm kiếm theo ý muốn .

### Using analyzers

Quá trình phân tích diễn ra tại 2 nơi : trong suốt quá trình indexing và khi sử dụng QueryParser.

Ví dụ một quá trình phân tích nhóm từ: “The quick brown fox jumped over the lazy dogs” và “XY&Z Comporation – xyz@example.com”.Với từng token được đưa ra giữa các dấu [] để tạo sự phân chia rõ ràng. Điều quan trọng nhất là các terms indexed thì là các terms có thể dùng để tìm kiếm.



Lucene không tạo ra kết quả phân tích có thể nhận thấy được đối với khách hàng (*end user*). Terms được tách ra từ đoạn văn bản nguyên bản để indexing và tìm phù hợp trong suốt trong quá trình searching. Khi searching với QueryParser quá trình phân tích diễn ra trở lại để đảm bảo kết quả phù hợp tốt nhất.

#### Indexing analysis

Trong suốt quá trình index , một Analyzer được nắm giữ từ IndexWriter.Với từng tokenized field của từng tài liệu được index sử dụng *IndexWriter* được sử dụng analyzer tùy trường hợp.Có hai loại Field đặc biệt được sử dụng để tạo tokennized là Text và Unstored. Khi index, có thể lựa chọn analyzer lúc IndexWriter hoặc theo từng cấp per-Document.

Field.KeyWord indexed field không phải là tokenized.Nó là một terms đơn lẻ với các giá trị đúng như một bộ cung cấp

#### QueryParser analysis

Analyzer là chìa khóa để terms indexed. Vì vậy phải cần chắc chắn rằng khi truy vấn phải đưa được chính xác terms indexed hợp trong tài liệu tìm kiếm hợp lệ. Khi sử dụng hàm API-created truy vấn như là TermQuery, người phát triển phần mềm cần phải chắc chắn đảm bảo rằng terms được sử dụng phải tương ứng khi indexed. QueryParser sử dụng analyzer để có thể làm tốt nhất công việc tìm terms tương ứng khi indexed. QueryParser phân tích chia các mẫu của biểu thức khi sử dụng câu lệnh:

Query query = QueryParser.parse(expression,”contents”,analyzer);

Biểu thức bao gồm các toán tử, các loại dấu ngoặc đơn, kép , các biểu thức cú pháp đặc biệt khác để biểu thị xếp loại , fuzzy searches.

#### Parsing versus analysis : khi một analyzer không phù hợp

Điều quan trọng của analyzers là khi sử dụng bên trong cho việc đánh dấu thành tokenized. Các tài liệu kiểu HTML, Microsoft Word, XML,… chứa các meta-dat như author, title, last modified date, potentially,… Khi indexing các tài liệu như kiểu meta-data có thể bị tách rời và indexed tạo thành fileds tách rời nhau.

Analyzers được sử dụng để phân tích không thể giúp được gì cho field separation vì các dãy thì được thỏa thuận với các field đơn tại một thời điểm.Trong những trường hợp này dữ liệu cần được parsed hoặc tiền xử lý , chuyển các separate blocks tách rời của đoạn text thành từng field (Chapter 7 sẽ nói rõ phần này).

### Analyzing the analyzer

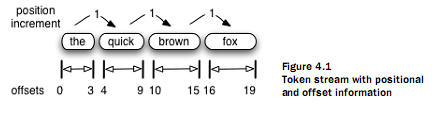
Điều quan trọng để xây dựng một bộ analyzer là phải biết rõ kiến trúc và xây dựng blocks provied, sử dụng Lucene đánh giá đúng mức và phải hiểu rõ bộ phân tích của Lucene, phải biết mở rộng và sửa lại một vài chỗ chưa hoàn hảo.

Analyzer là lớp cơ bản.Trong phần này sẽ nói rõ về từng thành phần chính được sử dụng bởi analyzers, bao gồm cả Token và TokenStream .

#### What’s in a token?

Một stream của tokens là đầu ra cơ bản của quá trình phân tích. Trong suốt quá trình indexing , fields được chỉ định cho tokenization thì được xử lý với analyzer, với mỗi token thì được viết ra trở thành index như là một term.

Ví dụ , khi analyzer đoạn text “the quick browm fox”. Với từng token trình bày các từ riêng biệt của đoạn text. Một token mang theo một giá trị của đoạn text và một vài giá trị sau khi dữ liệu được phân tích đó là offsets của điểm bắt đầu và kết thúc trong đoạn text ban đầu, loại token, sự độ tăng về vị trí (*position increment)* .



Start Offset là vị trí của ký tự đầu tiên trong đoạn text ban đầu được xuất hiện trong vị trí bắt đầu của token, end offset là vị trí của ký tự cuối cùng của token text. Loại token có thể là một “ *String”* , mặc định là “*Word*”, có thể điều chỉnh lại được quá trình lọc token nếu muốn. Trong đoạn text được tokenize , vị trí liên qua đến token trước đó thì được ghi lại như giá trị *position increment.*

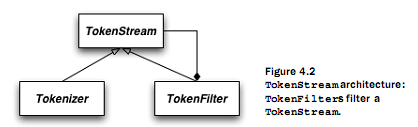
**Tokens into terms :** là đoạn text sau khi được analyzer trong suốt quá trình indexing, từng token được đặt đến index như một term. Position increment thì chỉ được thêm kết hợp meta-datavowis token được mang theo để index. Star offset và end offset là loại token thì bị loại bỏ , chúng chỉ được sử dụng trong quá trình phân tích.

**Position increments:** Là giá trị ghi lại sự tăng lên về vị trí của token hiện tại với token kế trước nó. Thông thường giá trị của nó là 1, chỉ ra với từng từ thì chỉ được chứ duy nhất và vị trí kế tiếp trong field. Vị trí này có giá trị lớn hơn 1 khi có một kẽ hở , từ tại vị trí đó trong quá trình index đã bị loại bỏ, ví dụ từ đó nằm trong tập hợp stop-word. Một token với một giá trị position increments là 0 khi nó có cùng vị trí như token trước nó. Analyzer xen vào một từ bí danh (*word aliases*).

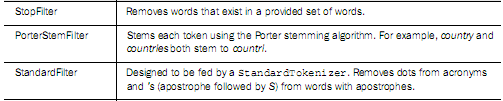
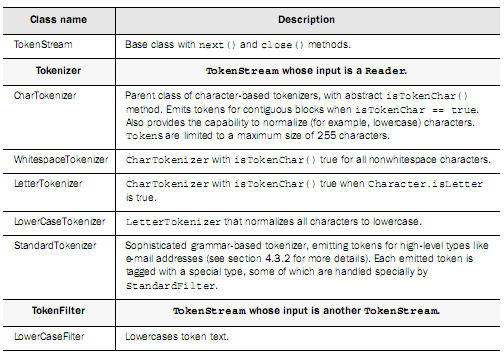
#### TokenStreams uncensored

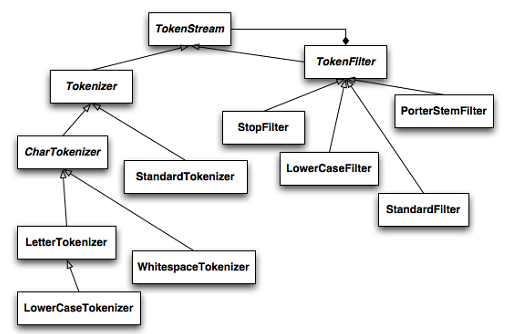
Có 2 loại TokenStream khác nhau : Tokenizer và TokenFilter. Tokenizer là liên kết của các ký tự riêng lẽ (*characters*), TokenFilter là liên kết của các từ (*words*).

Tokenizer là một TokenStream mà có tokenizes đưa vào từ một Reader, khi indexing một String trong Field.Text(String, String) hoặc Field.UnStore(String, String). Còn với loại TokenFilter thì được kết nối các TokenStream con lại . với từng TokenStream thì được xử lý kiểm tra xem là Tokenizer hay là TokenFilter để xử lý tiếp. TokenStream được thiết kế như mẫu Composite trong Design Pattern.

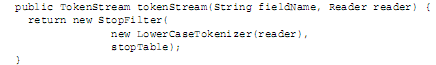


Các hàm bên trong:



Minh họa tổng quát, ở đây trung tâm là StopAnalyzer. 

Ví dụ một đoạn code trong StopAnalyzer:



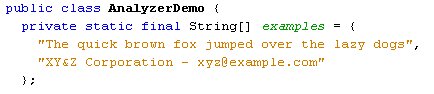
Trong StopAnalyzer này thì , LowerCaseTokenizer là đầu vào cung cấp cho StopFilter. LowerCaseTokenizer cho ra các token từ đoạn text ban đầu. Các lý tự không phải là chữ cái hoặc các chữ cái nằm trong Stop-Word list trở thành đường biên để tạo token và được loại ra ngoài.

Buffering là đặc trưng chung cần thiết trong TokenStream để nó có thể thực thi được. Low-Level tokenizers để buffer chứa characters từ form tokens khi chia được sự ngăn cách tại các vùng biên dựa vào whitespace, nonletter characters.

#### Visualizing analyzers

Điều quan trọng để hiểu được sự khác nhau của các loại analyzer với dữ liệu đưa vào. Có 4 bộ analyzers , có thể thấy được sự khác nhau khi sử dụng AnalyzerDemo , trong hàm này đưa vào 2 đoạn text để kiểm tra 4 bộ phân tích. Đầu ra là kết quả sau khi được phân tích và các term được trình bày tách biệt thông qua dấu [] , tức là kết quả sau khi được indexing.

Hai đoạn text được đưa vào phân tích:

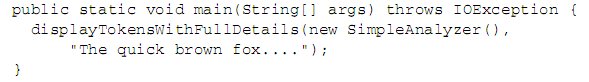


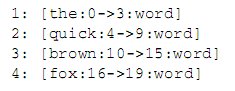
Kết quả:



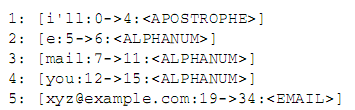
Một vài đặc điểm quan trọng của 4 bộ phân tích có trong Lucene:

* WhitespaceAnalyzer không thực hiện lowercase(chuyển tất cả sang ký tự thường ) , vẫn giữ dấu “-“ , loại bỏ tất cả các khoảng trắng, và dựa vào các khoảng trắng để làm đường biên phân chia tokenize.
* SimpleAnalyzer thực hiện lowercase, vẫn giữ các từ nằm trong danh sách stop-word, dùng các ký tự không phải là các chữ cái alphabetic để làm đường biên phân chia tokenize.
* Cả SimpleAnalyzer và StopAnalyzer đều làm hỏng tên của tên các công ty như ví dụ trên (xy&z bị rã ra thành [xy] [z] , loại bỏ ký hiêu &) .
* StopAnalyzer và StandarAnalyzer loại bỏ các từ nằm trong Stop-Word ví dụ như “*the*” ở ví dụ trên.
* StandarAnalyzer vẫn giữ được tên của công ty và thực hiện chức năng lowercase , loại bỏ dấu “-” , giữ được địa chỉ email (*xyz@example.com*).

**Looking inside tokens** TokenStream có thể tạo ra các token , TokenFilters có thể truy cập dữ meta-data . Thông qua ví dụ sau để giải thích cho việc truy cập dữ liệu data-meta :và kết quả đưa ra:

****

**What good are start and end offsets?**  Cả hai giá trị này không được sử dụng trong Lucene. Trong phần thảo luận section 8.7 về term highlighter sử dụng một TokenStream và kết quả là Tokens đưa ra ngoài để indexing xác định nơi nào a block của text bắt đầu và kết thúc highlighting, và đưa ra những từ trong kết quả tìm kiếm được cho là nổi bật nhất.

**Token-type usefulness** Hiển thị token-type để tạo sự riêng biệt trong từng loại của tokens. Ví dụ : **** kết quả đưa ra: 

#### Filtering order can be important:

Phần này chủ yếu sử dụng ví dụ stop-word để loại bỏ các từ, ký tự không quan trọng trong dữ liệu đưa vào, tạo ra một bộ lọc làm tinh dữ liệu.

### Using the built-in analyzers

Lucene gồm có một vài bộ phân tích chính như : WhiteSpaceAnalyzer, SimpleAnalyzer, StopAnalyzer , standardAnalyzer và có hỗ trợ 2 ngôn ngữ đặc biệt : RussianAnalyzer, GermanAnalyzer. Tìm hiểu rõ StopAnalyzer và StandardAnalyzer trong phần này.

#### StopAnalyzer

StopAnalyzer chia các từ cơ bản trong dữ liệu đưa vào và lowercasing , còn removes những stop words. Đưa vào các từ English thuộc stopword vào trong StopAnalyzer , ví dụ như list các từ : 

Bộ phân tích sau khi nhận dữ liệu đưa vào, rút gọn các từ chung được sử dụng trong English, ví dụ như *don’t , can’t, it’s.* Trước khi loại bỏ các từ có trong stop-word, StopAnalyzer giữ các ký tự lần lượt với nhau, ngăn chia tại các ký tự không phải là chữ cái bao gồm cả dấu “ ’ ” ( ví dụ như I’ll ).loại bỏ các ký tự như “s”, “t” đứng riêng lẻ, nó có khả năng phán đoán để loại bỏ các từ vô nghĩa.

Stop-word khi loại bỏ nảy sinh một số vấn đề như : làm gì với những lỗ hổng sau khi một số từ được loại bỏ ? Giả định bạn index “one is not enough”.Các token được tạo ra từ StopAnalyzer là [one] [enough] với “is” và “not” thì bị loại bỏ.Vì vậy kết quả sẽ đưa ra chính xác nếu như indexed có chứa dữ liệu “one enough”.Nếu như khi dữ dụng Query-Parser với StopAnalyzer, câu này sẽ dduwwocj match với các nhóm từ như “one enough”,”one is enough”,”one but not enought”, ”one is not enought”. Khi sử dụng QueryParser để phân tích câu trên, khi câu được chuyển sang các trường hợp khác nhau và khi đưa về trường hợp “one enough” thì được match với terms indexde. Đây chính là một “hole” trong topic này. Điều này sẽ được thảo luận kỹ trong phần 4.7.3.

Khi loại bỏ các từ thuộc list stop-word thì có thể ngữ nghĩa trong câu sẽ bị biến đổi, khả năng này có thể bị xảy ra nhưng cũng tùy vào ngữ cảnh mà nó mà tầm ảnh hưởng sẽ nhỏ lại, phụ thuộc vào việc sử dụng Lucene trong hoàn cảnh nào để tìm kiếm ý nghĩa của từ trong chương trình.

#### StandardAnalyzwer

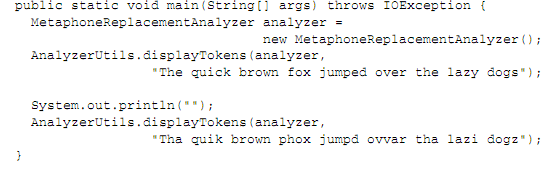
StandardAnalyzer là bộ phân tích chung nhất , nền tảng gắn liền với analyzer trong Lucene. Tokenizing thông minh với các loại từ vựng, chữ cái , viết tắt của các chữ đầu dòng, tên công ty, địa chỉ email, tên miền , số, serial number. IP address, CJK (Chinese, Japanese, Korean) character. Loại bỏ các từ nằm trong stopword , vì vậy StandardAnalyzer thường là lựa chọn đầu tiên.

### Dealing with keyword fields

Dễ dàng index một keyword bằng cách sử dụng Field.Keyword.Trong tìm kiếm media phần này không sử dụng, chủ yếu dùng cho tìm kiếm tin tức.

### “Sounds like” querying

Xử lý những từ đồng âm, khi người dùng tìm kiếm không ghi đúng từ cần tìm mà ghi lại từ đồng âm để tìm kiếm . Ví dụ như chuỗi ký tự đưa vào để tìm kiếm “Kool kat” tức là họ muốn tìm “cool cat”. Lucene cũng hỗ trợ để lập trình viên giải quyết được trường hợp này.

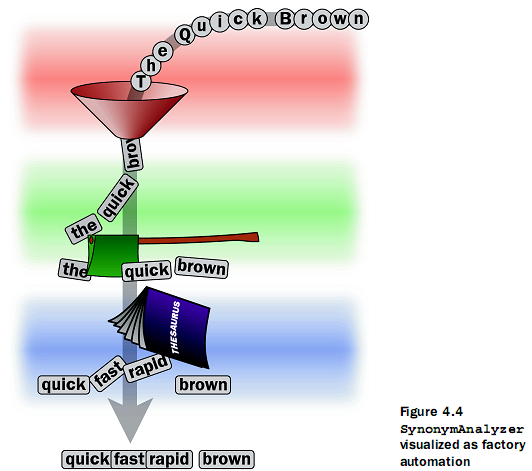


Kết quả đưa ra dữ liệu theo kiểu Metaphone Encoder :



“sounds-like” được thiết kế đặc trưng cho search engines cho trẻ em rất hữu dụng, hầu hết những sounds-like được thấy đều là những từ được viết sai chính tả , vì vậy nó cần thiết cho việc tra cứu hỏi ý kiến người sử dụng hiệu chỉnh lại từ khóa tìm kiếm khi cần thiết.

### Synonyms, aliases, and words that mean the same



Analyzer thêm vào các từ đồng nghĩa trong chuỗi token được đưa ra ngoài , nhưng các từ được thay thế thì được đặt ở cùng vị trí trong từ gốc.

### Stemming analysis

Những lỗ hổng được dời đi ở những nơi stop words được loại bỏ khi thực hiện chức năng phân tích dùng PositionalStopFiler.

Những từ có dạng khác biệt của một từ thì được chuyển về dạng chung nhất, thường hay gặp trong tiếng Anh ví dụ như : breathe, breathes, breathing, breathed thông qua Porter stemmer thì đều được chuyển lại thành breath.

### Language analysis issues

Việc liên kết các ngôn ngữ khác nhau trong Lucene là một điều thú vị và có rất nhiều vấn đề phức tạp. Làm thế nào một đoạn text với các ngôn ngữ khác nhau được index và sau đó được tìm lại được? vì vậy sẽ có rất nhiều vấn đề mà người phát triển phần mềm cần xem xét.

Rào cản đầu tiên đó là các ký tự được mã hóa (encoding) , trong suốt quá trình phân tích các ngôn ngữ khác nhau thì có tập hợp stop-words khác nhau và có một thuật toán stremming algorithms duy nhất. Lucene có hỗ trợ một số bộ phân tích được xây dựng hỗ trợ cho developer như Tokenizers và TokenStreams có trong SandBox.

Vấn đề ngôn ngữ cần được xem xét trong luận văn đó là ngôn ngữ Unicode và encodings. Lucene có chứa tất cả các ký tự trong UTF-8 encoding chuẩn, đó chính là điều kiện thuận lợi để đưa vào dữ liệu encoding cho bộ phân tích. Khi đọc một tập tin HTML hoặc XML từ HTTP server , encoding chính là vấn đề gặp phải, các ký tự phức tạp gây khó khăn cho quá trình phân tích.

Ngoài ngôn ngữ English , Lucene còn hỗ trợ rất nhiều ngôn ngữ khác cho bộ phân tích .Ví dụ như GermanAnalyzer và RussianAnalyzer , ngoài ra còn có bộ SnowballAnalyzer hỗ trợ cho hầu hết các ngôn ngữ European . Bộ phân tích cho các ngôn ngữ tượng hình điển hình như các nước : Trung Quốc, Hàn Quốc, Nhật Bản (CJK ) . Chỉ có duy nhất StandardAnalyzer đi kèm trong bộ Analyzer là hữu dụng cho tất cả các ngôn ngữ châu Á. Tuy nhiên , có 2 bộ Analyzer trong Lucene Sandbox phù hợp cho các ngôn ngữ châu Á, húng thì không được đính kèm trong core Lucene .

Một vấn đề đặc biệt nữa cần quan tâm đó là dữ liệu đưa vào gồm nhiều ngôn ngữ.Khi bạn indexing tài liệu chứa nhiều ngôn ngữ từ một single index, sử dụng per-Document analyzer là thích hợp, tức là dung từng bộ phân tích cho từng tài liệu cho từng ngôn ngữ. Nếu vẫn phải bắt buộc thêm field dữ liệu đa ngôn ngữ, thì field đó cần sử dụng bộ lọc cho kết quả tìm kiếm , hoặc hiển thị mục đích trong suốt quá trình khôi phục

Bộ phân tích là một khía cạnh trong Lucene có nhiều vấn đề cần được quan tâm và nỗ lực giải quyết một cách xứng đáng.Sử dụng StandartAnalyzer là bí quyết để dùng cho quá trình indexing analysis cho nhiều ứng dụng chung , nhưng điều quan trọng trong quá trình phân tích , người sử dụng cung cấp một đoạn text tìm kiếm và thường có thể sẽ gây ra những kết quả nhầm lẫn , ví dụ khi tìm kiếm “to be or not to be” sẽ không cho một kết quả nào vì trong lúc quá trình analyzer các từ nằm trong bộ lọc đã loại bỏ các từ nằm trong danh sách StopWord.

Khi thay đổi bộ phân tích thì điều bắt buộc phải rebuild lại index và cũng sử dụng bộ phân trích mới cho tất cả tài liệu đã được phân tích trước đó.

# CÁC KĨ THUẬT TÌM KIẾM NÂNG CAO

Như đã đề cập,đã có nhiều ứng dụng cài đặt chức năng tìm kiếm dựa vào thư viện Lucene API.Nhiều ứng dụng đã xây dựng được những bộ tìm kiếm hoàn thiện với nhiều chức năng-trong đó phải kể đến Nutch của cùng tác giả,và để có thể làm được điều này,ta cần tìm hiểu sâu hơn những khả năng khác đã được xây dựng trong Lucene

### Sắp xếp kết quả trả về

Từ Lucene 1.4 các kết quả trả về được sắp xếp theo thứ tự giảm của độ ưu tiên (Score).nghĩa là các tài liệu nào có kết quả chính xác hơn sẽ xuất hiện ở đầu.Tuy nhiên với những phương thức chồng (overloaded) của sẻarch ta hoàn toàn có thể sắp xếp theo nhiều cách khác ,bao gồm cả việc sắp xếp các trường (Field) tùy tham số truyền vào.

Trong những trường hợp sau: ta đều sử dụng phương thức search và trả ra kết quả là collection Hits-chứa kết quả đã được sắp xếp .Ở đây ta chỉ cần quan tâm tới đối tượng Sort truyền vào ở dạng nào mà thôi.



### Sắp xếp theo độ chính xác

Đây là cách sắp xếp thường thấy và là mặc định trong phương thức search (Query) của đối tượng IndexSearcher.để sắp xếp theo kiểu này tham số truyền vào của Sort có thể là

* Search(query,null)
* Search(query,Sort.RELEVANCE)
* Search(query,new Sort())

### Sắp xếp theo độ thứ tự được đánh chỉ mục(index order)

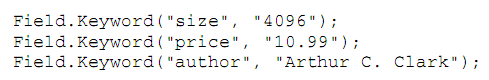
Thứ tự chỉ mục gọi là Id,tức là vị trí của tài liệu được lưu trong chỉ mục tùy vào loại truy vấn Query.chú ý rằng thứ tự của tài liệu trong chỉ mục không phải là độ ưu tiên-Score.Sắp xếp kiểu này Id sẽ được sắp tăng dần với tham số:

* Search(query,Sort.INDEXORDER)

### Sắp xếp theo Trường (Field)

Để săp xếp theo Field,ta cần chú ý một nguyên tắc là:

* Field đó phải được indexed nhưng không được tokened,chẳng hạn như là Field.Keyword
* Giá trị của Field đó phải được chuyển thành kiểu Integer,Floats hoặc là Strings,ví dụ như:



Khi đó,cần sắp xếp theo Field nào ta chỉ cần khởi tạo đối tượng Sort theo Field đã index là được,như ví dụ trên ta muốn sort theo size ta cần:

* Search(query,new Sort(“size”))

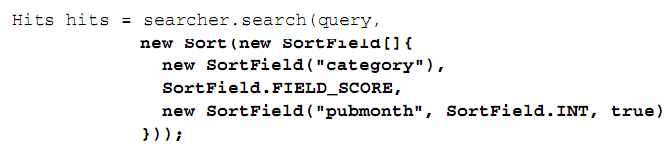
### Đảo ngược thứ tự đánh chỉ mục

Mặc nhiên,thứ tự đánh chỉ mục là giảm dần theo độ chính xác, và tăng dần các trường khác.Tuy nhiên ta hoàn toàn có thể đảo ngược thứ tự này theo từng trường ,chẳng hạn ví dụ sau sắp theo độ giảm của trường tên là “size”-tức là tài liệu nào có size lớn hơn sẽ xuất hiện ở đầu,bằng cách sau:

* Search(query,new Sort(“size”,true))

### Sắp xếp theo nhiều trường khác nhau(multiple fields)

Để thực hiện việc sắp xếp theo nhiều trường khác nhau theo tiêu chí nào đó,ta sử dụng một mảng SortFields và add các Field cần sắp xếp vào trong mảng đó,



Trong ví dụ trên,ta sắp xếp theo độ giảm độ ưu tiên của field “category”,đồng thời nếu các cuốn sách trong cùng “category” được sắp giảm theo năm xuât bản.

SortField có thể nhận các hằng số :

|  |  |
| --- | --- |
| SortField | Ý nghĩa |
| FIELD\_SCORE | kiểu sắp xếp theo độ chính xác |
| FIELD\_DOC | Sắp xếp theo ID của tài liệu |
| FIELD\_AUTO | Sắp xếp theo thứ tự Alphabate của tên Field |
| FIELD\_STRING , FIELD\_INT, FIELD\_FLOAT | Tùy vào từng field nhận kiểu dữ liệu nào mà ta sắp xếp theo kiểu tương ứng |

# EXTENFING SEARCH

Các nội dung chính được đưa ra và giải quyết trong chương này:

* Tạo một “custom sort”.
* Sử dụng HitCollector.
* Customizing QueryParser.
* Thực hiện kiểm chứng.

## Using a custom sort method

Phần này tạo ra một phương thức sắp xếp tùy chọn theo ý muốn ,ví dụ như ngoài những kiểu sắp xếp cơ bản như theo điểm số (score ), ID , giá trị của fields (field values) Lucene còn giúp bạn thi hành một custom sorting được cung cấp từ SortComparatorSource . Thay đổi làm lại phương thức sắp xếp để nó có thể thi hành một cách hữu dụng nhất khi sắp xếp theo các tiêu chuẩn cơ bản không thể định rõ được trong suốt quá trình tạo chỉ mục (indexing).

Để giải thích rõ hơn về phần này cuốn sách “Lucene in action” đã đưa ra một ví dụ tìm kiếm sắp xếp theo chuẩn mực của các điểm từ gần cho đến xa dần so với một địa điểm cho trước dựa vào các tọa độ của các địa điểm cho trước.Khi thực hiện các giải thuật để tính toán khoảng cách tuyệt đối của các điểm với điểm được chọn làm cột mốc ban đầu thì vẫn còn một số thông tin lạ thường không đúng. Tuy nhiên , lower-level API giúp chúng ta truy cập sâu hơn vào giá trị được sử dụng để sắp xếp.

Ngoài các phương thức trong IndexSearcher.search còn có các phương thức lower-level được sử dụng bên trong Lucene và nó không được sử dụng ra bên ngoài. Những trường hợp ngoại lệ được đi sâu truy cập vào bên trong các thuộc tính để sắp xếp. Lower-level API yêu cầu chúng ta chỉ định có bao nhiêu kết quả giá trị trả về mà chúng ta mong muốn, đều này khác với việc yêu cầu phương thức Hits-returning. Lower-level thì thường được dùng trong các trường hợp custom sort values.

# CÁC KĨ THUẬT TÌM KIẾM NÂNG CAO

Như đã đề cập,đã có nhiều ứng dụng cài đặt chức năng tìm kiếm dựa vào thư viện Lucene API.Nhiều ứng dụng đã xây dựng được những bộ tìm kiếm hoàn thiện với nhiều chức năng-trong đó phải kể đến Nutch của cùng tác giả,và để có thể làm được điều này,ta cần tìm hiểu sâu hơn những khả năng khác đã được xây dựng trong Lucene

### Sắp xếp kết quả trả về

Từ Lucene 1.4 các kết quả trả về được sắp xếp theo thứ tự giảm của độ ưu tiên (Score).nghĩa là các tài liệu nào có kết quả chính xác hơn sẽ xuất hiện ở đầu.Tuy nhiên với những phương thức chồng (overloaded) của sẻarch ta hoàn toàn có thể sắp xếp theo nhiều cách khác ,bao gồm cả việc sắp xếp các trường (Field) tùy tham số truyền vào.

Trong những trường hợp sau: ta đều sử dụng phương thức search và trả ra kết quả là collection Hits-chứa kết quả đã được sắp xếp .Ở đây ta chỉ cần quan tâm tới đối tượng Sort truyền vào ở dạng nào mà thôi.



### Sắp xếp theo độ chính xác

Đây là cách sắp xếp thường thấy và là mặc định trong phương thức search (Query) của đối tượng IndexSearcher.để sắp xếp theo kiểu này tham số truyền vào của Sort có thể là

* Search(query,null)
* Search(query,Sort.RELEVANCE)
* Search(query,new Sort())

### Sắp xếp theo độ thứ tự được đánh chỉ mục(index order)

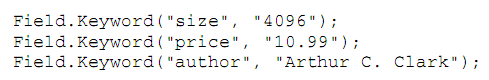
Thứ tự chỉ mục gọi là Id,tức là vị trí của tài liệu được lưu trong chỉ mục tùy vào loại truy vấn Query.chú ý rằng thứ tự của tài liệu trong chỉ mục không phải là độ ưu tiên-Score.Sắp xếp kiểu này Id sẽ được sắp tăng dần với tham số:

* Search(query,Sort.INDEXORDER)

### Sắp xếp theo Trường (Field)

Để săp xếp theo Field,ta cần chú ý một nguyên tắc là:

* Field đó phải được indexed nhưng không được tokened,chẳng hạn như là Field.Keyword
* Giá trị của Field đó phải được chuyển thành kiểu Integer,Floats hoặc là Strings,ví dụ như:



Khi đó,cần sắp xếp theo Field nào ta chỉ cần khởi tạo đối tượng Sort theo Field đã index là được,như ví dụ trên ta muốn sort theo size ta cần:

* Search(query,new Sort(“size”))

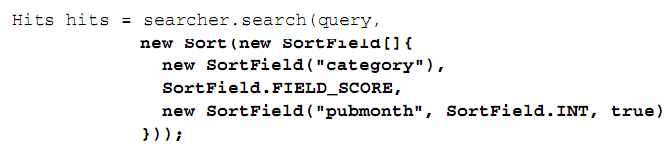
### Đảo ngược thứ tự đánh chỉ mục

Mặc nhiên,thứ tự đánh chỉ mục là giảm dần theo độ chính xác, và tăng dần các trường khác.Tuy nhiên ta hoàn toàn có thể đảo ngược thứ tự này theo từng trường ,chẳng hạn ví dụ sau sắp theo độ giảm của trường tên là “size”-tức là tài liệu nào có size lớn hơn sẽ xuất hiện ở đầu,bằng cách sau:

* Search(query,new Sort(“size”,true))

### Sắp xếp theo nhiều trường khác nhau(multiple fields)

Để thực hiện việc sắp xếp theo nhiều trường khác nhau theo tiêu chí nào đó,ta sử dụng một mảng SortFields và add các Field cần sắp xếp vào trong mảng đó,



Trong ví dụ trên,ta sắp xếp theo độ giảm độ ưu tiên của field “category”,đồng thời nếu các cuốn sách trong cùng “category” được sắp giảm theo năm xuât bản.

SortField có thể nhận các hằng số :

|  |  |
| --- | --- |
| SortField | Ý nghĩa |
| FIELD\_SCORE | kiểu sắp xếp theo độ chính xác |
| FIELD\_DOC | Sắp xếp theo ID của tài liệu |
| FIELD\_AUTO | Sắp xếp theo thứ tự Alphabate của tên Field |
| FIELD\_STRING , FIELD\_INT, FIELD\_FLOAT | Tùy vào từng field nhận kiểu dữ liệu nào mà ta sắp xếp theo kiểu tương ứng |

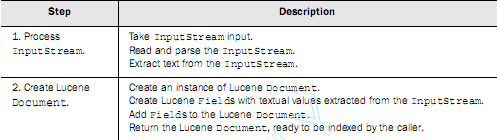
# PARSING COMMON DOCUMENT FORMAT

* Parsing XML sử dụng SAX 2.0 API và Jakarta Commons Digester
* Parsing PDF documents với PDFBox
* Parsing HTML sử dụng Jtidy và NekoHTML
* Parsing Microsoft Word documents với Jakarta POI và TextMining.org API
* Parsing RTF documents sử dụng parser gắn kèm trong JDK
* Tạo một document indexing framework và chương trình ứng dụng

### Handling rich-text document

Trong phần này sẽ giúp tạo một framework nhỏ để có thể sử dụng index một document chung nhất được tìm thấy trong môi trường văn phòng hay internet .

Để tạo một DocumentHandler chung thì sử dụng InputStream để làm đối số đầu vào bởi vì hầu hết các công cụ hỗ trợ để tách phần text ra từ các file tài liệu định dạng khác nhau đều đưa ra InputStream. Khi thi hành thì kết quả trả về sẽ là một class Document bao gồm một hoặc nhiều field, bởi vì các định dạng khác nhau thì được chứa trong các meta-dât khác nhau , ví dụ đặc trưng của file định dạng HTML có titles còn XML thì không có vì vậy HTML DocumentHandler có thể trả ra một Document với một field là title nhưng XML thì không có . Tất cả các lỗi khi sử lý được thi hành thông qua DocumentHandlerException .



Khi trích ra được text thông qua parser tài liệu đầu vào sẽ được đưa vào hàng đợi để indexing cho Lucene Documents . Bước indexing thì đồng nhất cho tất cả các loại document , trong quá trình này có thể xảy ra quá trình trùng lặp trong quá trình parser, trường hợp này đã xử lý thông qua một gói frameword để thêm fields chung cho tất cả các tài liệu khi cần (Ví dụ những phần dễ trùng lặp như : last modifield date, file system path, URL , …).

### Indexing XML

Đề lấy những mẩu thông tin trong XML document chuyển thành một Lucen Document thì dùng SAX API hoặc dùng Jakarta Common Digest. Chúng ta có thể index những mẩu đó đó với Lucene.

* Parsing và indexing sử dụng SAX :Cung cấp các hàm API gồm các phương thức hỗ trợ để gọi xử lý các sự kiện để làm việc với định dạng XML. Một ứng dụng dùng SAX được phổ biến rộng rãi đó là Xerces2Java Parser , nó được phát triển dựa trên nền Apache XML project , được lưu trữ tại trang web <http://xml.apache.org/xerces2-j/index.htm> trên phiên bản SAX API 2.0 .
* Parsing và indexing sử dụng Digester: Digester được lưu trữ tại <http://jakarta.apache.org/commons/digester> nó là một subproject của Jakarta Common project.

### Indexing một DPF document

PDFBox thì miễn phí , thư viện mã nguồn mở được viết bởi ben Litchfield, bạn có thể tìm nó tại <http://www.pdfbox.org> . Hiện có một vài công cụ có khả năng rút trích có đoạn text dữ liệu từ file pdf , nhưng PDFBox thì được phổ biến nhất.

Gói org.pdfbox.searchengine.com của PDFBox chứa 2 lớp IndexField và LucenePDFDocument.

### Indexing an HTML document

HTML có ở mọi nơi , hầu hết các tài liệu web đều được lưu trữ ở dạng HTML Chuyển HTML không phải đơn giản vì hiện tại còn nhiều website vẫn không phù hợp với tiêu chuẩn cuối cùng của W3C cho XHTML .Có thể dùng Jtidy hoặc NekoHTML để chuyển đổi định dạng HTML.

Tidy là một phiên bản cũ dùng để chuyển đổi định dạng HTML được viết bằng ngôn ngữ C bởi tác giả Dave Raggett nhưng dự án đã dừng lại năm 2000. Có thể tìm thấy mã nguồn mở của Tidy tại <http://tidy.sourceforge.net> . Jtidy là một port của Tidy được viết bởi Andy Quick , có thể tìm tại <http://jtidy.sourceforge.net>

NekoHTML chuyển đổi HTML và truy xuất dữ liệu dùng phương pháp sử dụng cân bằng tag trong XML chuẩn. NekoHTML là một phần của tập hợp công cụ Cyber Neko cho XNI của Andy Clark. Có thể tìm thấy tại <http://www.apache.org/~andy/neko/doc/index.html>

### Indexing a Microsoft Word document

Hiện nay bộ soạn thảo Microsoft Word vẫn là phần mềm được sử dụng nhiều nhất trong soạn thảo văn phòng. Vì vậy việc thực hiện indexing được MS Word là một việc thiết thực cần phải làm , cần phải biết cách parse tài liệu MS Word để có thể tìm kiếm với Lucene. Nhưng sự khó khăn nhất gặp phải đó là MS Word là chương trình có bản quyền vì vậy format của nó cũng được giấu kín. Nhưng may thay , có một vài project mã nguồn mở có thể khắc phục được vấn đề này . Trong phần này có giới thiệu 2 project mã nguồn mở Jakarta POI và TextMining.org dùng để rút trích text ra.

#### Using POI

Poi là một Jakarta project , có thể tìm nó tại <http://jakarta.apache.org/poi> . Nó là project cung cấp các hàm API của java để thao tác với các dạng format cơ bản trên định dạng Microsoft’s OLE 2 Compound Document .

Các hàm trong lớp POI có thể lấy ra text của một file tài liệu MS Word , thêm nữa còn có thể sử lý với các meta-data như là summary , tên tác giả , … đi kèm theo file.

#### Using TextMining.org’s API

Các hàm API của TextMining.org còn có thể cung cấp một số khả năng khác nữa so với POI . Có một số phần các hàm của nó được thực hiện tốt hơn POI ví dụ như các hàm thư viện hỗ trợ có tỉ lệ thành công khi lấy text ra từ file MS Word , các thư viện hàm hỗ trợ rút trích text hỗ trợ các phiên bản Word 6/95 mà POI không hỗ trợ được,...

### Indexing an RTF document

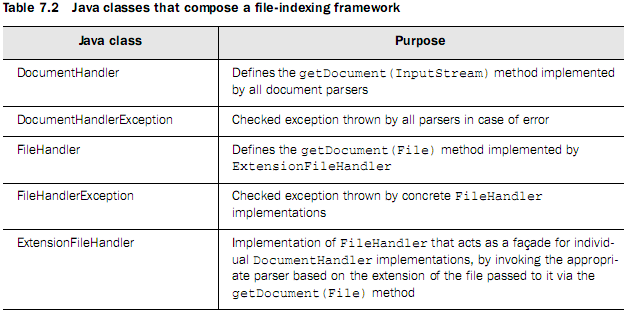
Với Rich Text Format (RTF) chúng ta có thể sử dụng bộ thư viện có sẵn trong gói Java chuẩn , chúng được ẩn trong javax.swing.text và javax.swing.rtf . Các text được lấy ra từ file RTF được chuyển đến Lucene trở thành như Field.Unstored . Cuối cùng DocumentHandler sẽ xử lý phần file plain-text .

### Indexing a plain-text document

Cuối cùng phần thực thi một DocumentHander cho plain-text , phần kết quả thì được tạo như một frameword cho parsing và indexing document cho các phần định dạng khác nhau.

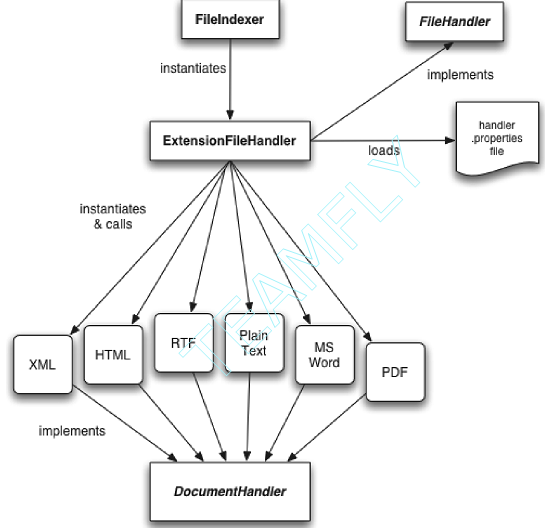
### Creating a document – handling framwork

Phần này nói về sự kết nối của các phần phức tạp khác nhau thành một khối thống nhất , là thi hành parsing của một vài định dạng dữ liệu khác nhau bởi một framwork duy nhất .

Cấu trúc gồm có DocumentHandle interface và kèm theo nó là FileHandlerException và nó thực thi với lớp ExtensionFileHandle. Bảng tóm tắt Framwork components : 

##### FileHandler interface

FileHandler là một interface đơn giản giống như là DocumentHandle . Tuy nhiên , điểm khác đó là DocumentHandle dùng giống như InputStream để làm loại dữ liệu đưa vào , còn FileHandler interface định nghĩa File là loại dữ liệu đưa vào ,điều đó sẽ làm việc dễ dàng hơn .



#### Extension FileHandler

ExtensionFileHandler chỉ thực thi với FileHandler interface và trả ra một Lucene Document chung . FileHandler thực thi với bất kỳ gói , nếu ngoại lệ sẽ được thông qua FileHandlerException .

ExtensionFileHandler chỉ thực thi trong FileHandler interface. Khi thực thi phương thức getDocument(File) sử dụng phần mở rộng của file để suy ra được loại file và gọi hàm thi hành parser thích hợp. Bởi vì tất phần thực thi parser đều thuộc phần chung DocumentHandler interface, ExtensionFileHandler có thể dò ra được đối tượng File đưa vào trong gói FileInputStream nó có thể biết được cần phải xử lý như thế nào.

#### FileIndex application

FileIndex có thể parse và index tất cả các định dạng dữ liệu .

#### Using Index

#### FileIndexer drawbacks and how to extend the framework

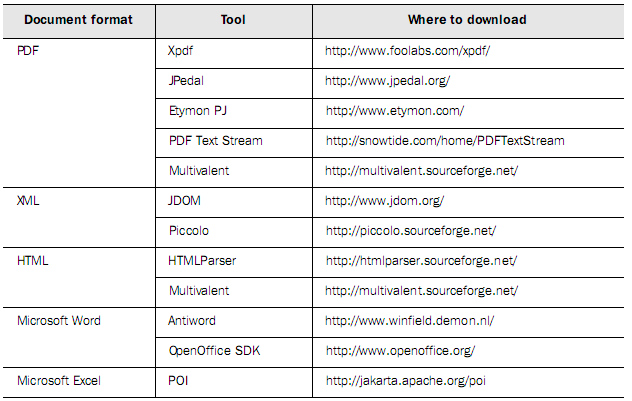
Framework bao có thể parser xử lý các loại file đưa vào như :XML, PDF, HTML, Microsoft Word, RTF , Plain text.

Nhưng nếu cần index và tạo ra file có thể tìm kiếm được với những loại file mà framwork không hỗ trợ thì có thể mở rộng framwork theo các bước sau:

* Viết một parser cho loại file mong muốn và thực thi DocumentHandler interface
* Thêm class parser đến file handler.properties , ánh xạ thích hợp với phần mở rộng của loại file mong muốn.
* Vẫn sử dụng FileIndexer .

### Other text-extraction tools

Trong phần này ta sẽ thấy sự hiện diện của các text được rút trích ra từ , phần indexing của các loại định dạng dữ liệu phổ biến nhất.



### Document-management systems and services

Thêm vào một số bộ thư viện hỗ trợ việc thi hành parser các loại dữ liệu và indexing bằng nhiều cách khác nữa , một vài gói phần mềm và services miễn phí có thể tin tưởng cho Lucene xử lý indexing :

* DocSearcher (<http://www.brownsite.net/docsearch.htm> ) có mô tả của tác giả: “DocSearcher sử dụng Open Source Lucene và POI Apache APIs tốt nhất cho mã nguồn mở Open Source PDF Box API cung ứng tìm kiếm TML, MS Word, MS Excel, RTF, PDF, Open Office (và Star Office) documents, và text documents.”
* Docco (<http://tockit.sourceforge.net/docco/index.html> ) thì nhỏ, theo dữ liệu tại trang chủ thì nó có thể xử lý một số định dạng dữ liệu sau: plain text, XML, HTML, PDF, Microsoft Word và Excel, OpenOffice, và StarOffice 6.0, có thể thực hiện tốt trên UNIX. Không thực hiện với RTF documents.
* SearchBlox (<http://www.searchblox.com/> ) là một tìm kiếm J2EE được triển khai dưới dạng chương trình web. Nó có thể index và search HTML, PDF, Word, Excel, và PowerPoint documents.
* Simpy (<http://www.simpy.com/> ) Một service miễn phí trực tuyến. Nó giúp lưu lại link trực tuyến cho dữ liệu của bạn dưới định dạng như HTML web pages; PDF, Microsoft Word, RTF documents; hay các định dạng khác. Bên cạnh meta-data bạn có thể đi sâu vào từng document, Simpy sẽ crawl và index the tất cả đầy đủ text trong document , đồng cho tìm kiếm trên bất cứ máy tính nào. Dữ liệu của bạn có thể che dấu hoặc chia sẻ trực truyến. Tất indexing và searching là sức mạnh của Lucene, và Nutch là phần cho end user.

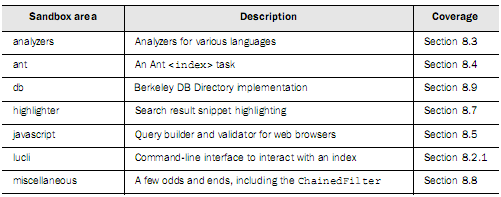
# TOOLS AND EXTENSIONS

Trong chương này giới thiệu những công cụ hỗ trợ index , browse hay query. Có một vài bộ phân tích (analyzer) đặc trưng cho nhiều ngôn ngữ có trong Sandbox của Lucene. Cân nhắc chú ý dùng những phần source code đi kèm theo phần thiết kế Lucene vì có cuốn sách khác nhau về Lucene có sự riêng biệt về các công cụ hỗ trợ được phát triển ngày càng nhiều và mới vì Lucene là mã nguồn mở.

### Playing in Lucene’s Sandbox

Ngày nay , các đóng góp cung cấp phát triển mã nguồn mở liên quan đến Lucene project đã và đang ngày càng tăng lên một cách nhanh chóng, vì vậy kho chứa Sandbox CVS đã được tạo ra.Sandbox liên tục được phát triển , đưa ra và phát triển những mẫu dự án cụ thể và thực hiện có định hướng rõ ràng. Khi cần những gì cần thiết với những phần liên quan đến Lucene có thể tìm kiếm trong Sandbox , sẽ có rất nhiều kết quả tìm thấy để hỗ trợ , và nó là mã nguồn mở nên bạn cũng có thể ý cung cấp chia sẽ mã nguồn mở liên quan đến Lucene.

Sau đây là những nội dung chính hiện có trong Sandbox tại thời điểm hiện tại viết cuốn Lucene in action:



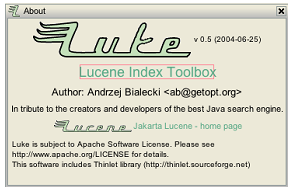


### Interacting with an index

Bạn có thể tự tạo ra một index hay bằng cách viết bằng code Java tương tác với Lucene áp dụng vào chương trình của bạn.Có một vài tiện ích đã được tạo ra cho bạn sử dụng tương tác với Lucene trong sử lý index:

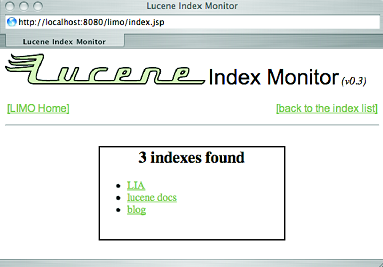
■ lucli (Lucene Command-Line Interface)—là một giao diện command line giúp truy vấn đặc biệt và kiểm tra index. Lucli được đóng góp từ Dror Matalon, nó cung cấp những dòng lệnh thực thi , cho phép xem lại những dòng lệnh đã thực thi, quay lại thời điểm thực thi những dòng lệnh trước đó.Lucli tương tác với một chuỗi hoạt động liên tiếp . Nó có 2 hạn chế đã được ghi chép lại nhưng nhìn chung thì vẫn không làm giảm sự hữ dụng của nó : phiên bản hiện tại nó cho sử dụng MultiFieldQueryParser cho tìm kiếm và hard-coded sử dụng StandardAnalyzer với parser.

■ Luke (Lucene Index Toolbox)—Một chương trình giao diện desktop với các tiện ích hay . Andrzej Bialecki đã tạo ra Luke (found at <http://www.getopt.org/luke/> ). Luke trực quan đối với người sử dụng cho index browser.



Tiện ích Luke cho phép hỗ trợ đối với những query đặc biệt , cung ứng những terms và structure từ index. Luke dần trở thành công cụ chuẩn cho phát triển Lucene, giúp user có thể sử dụng browsing và thử nghiệm nhanh. Được cài đặt từ gói file java , là Java WebStart nhưng được cài đặt trên localhost.

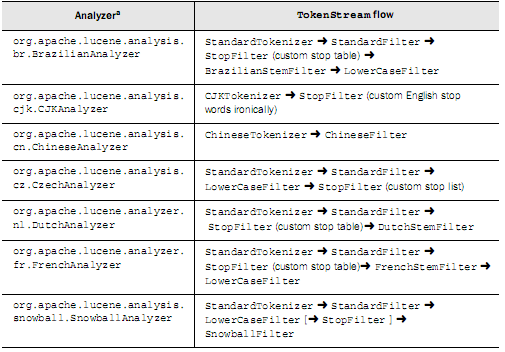
Có 5 tab trực quan trong Luke : Overview, Documents, Search, File, Plugins.Các chức năng được phân chia rõ ràng tường minh cho người dùng dễ dàng sử dụng. Ví dụ như trong tab Overview khi sử dụng cần điều chỉnh lại đường dẫn file index và những file được chọn để thực hiện index. Index có những điểm hay khi sử dụng Luke như cho phép thống kê “Top ranking terms”, có thể xem những tài liệu chứ 1 term được phân tích và hiển thị , … Khi Search cho hiện kết quả , tần xuất xuất hiện trong từng văn bản tìm được, những term được chứa ,… Ngoài ra còn cho hỗ trợ sử dụng những plug-in cho Luke cho sử dụng những chức năng mới.

■ LIMO (Lucene Index Monitor)—Giao diện web cho phép remote index browsing. Julien Nioche là tác giả của Lucene Index Monitor(LIMO) , link <http://limo.sourceforge.net/> , là một giao diện web để tương tác Lucene index . Từ link trên có thể tải source code và cài đặt cho localhost , có thể dùng Tomcat để thực thi source code , ngoài ra cần phải thêm thư viện liên quan từ Lucene , sử lại nội dung trong file limo/WEB-INF/web.xml cho phù hợp. Sau khi cài đặt xong sẽ chạy với localhost : <http://localhost:8080/limo/index.jsp> 

Phần sử dụng bạn có thể tự tìm hiểu .

### Analyzers, tokenizers, and TokenFilters,oh my

Có nhiều bộ Analyzer để tham khảo , nhưng Sandbox không thể làm thỏa mãn hết các yêu cầu theo từng vùng . Nó hỗ trợ một vài ngôn ngữ thông dụng cho bộ analyzer , bộ lọc , tokenizers và thuật toán trong Snowball của analyzer. Một số bộ phân tích được tóm lược trong bảng sau :



* SnowballAnalyzer

Có những điểm nổi bật như có thể tách ra như một phần driver của một tổng thể để phân chia cho các ngôn ngữ khác nhau. . Nếu trong project của bạn có nhu cầu về stremming , đầu tiên bạn có thể nghĩ ngay đến Snowball analyzer , chắc chắn bạn phải custom lại nó để có thể sử dụng theo nhu cầu của project của bạn.

* Obtaining the Sandbox analyzers

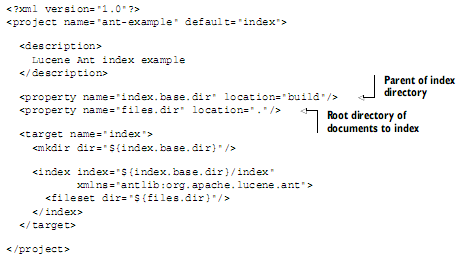
Tùy thuộc vào những cái bạn cần , bạn có thể sử dụng các gói bộ phân tích trong bộ thư viện để mượn những ý tưởng và tạo ra một bộ phân tích mới tùy ý.

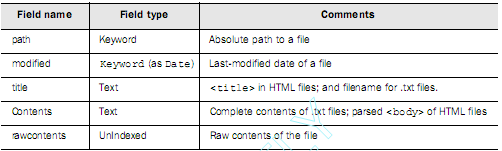
### Java Development with Ant and Lucene

Một điểm tự nhiên với Lucene đó là sự kết hợp các tài liệu indexing trong suốt quá trình build. Vì dữ liệu thường xuyên được thay đổi , vì vậy cần phải tìm kiếm dữ liệu trên những phiên bản mới nhất của dữ liệu được cập nhật. Java Development with Ant( Hatcher and Loughran, Man-

ning Publications, 2002), Erik created an Ant task to index a directory of file-based .Và code này đã được tăng lên và duy trì trong Sandbox.

* Using the <index> task

Ví dụ: 



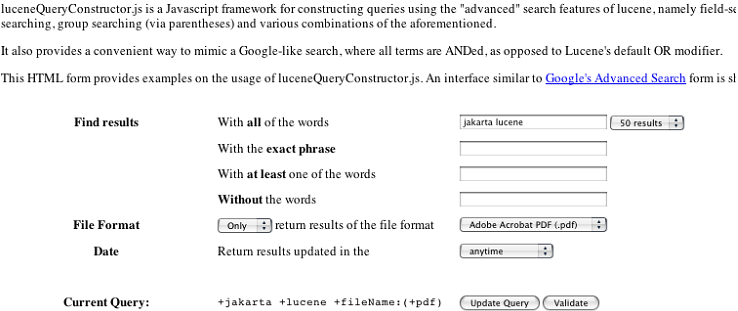
Mặc định những file xử lý làm .txt hoặc .html . Nếu như chỉ sử dụng mặc định xử lý trên những document thì không đủ , nhưng có thể custom document handler .

### JavaScript browser utilities

QueryParser là một tiện ích , nó dễ dàng đưa ra những textbox đơn giản đồng ý cho phép user gõ vào một truy vấn, nhưng nó không thân thiện khi sử dụng đối với user để có thể thấy được tùy chọn truy vấn riêng biệt từ field , như là lựa chọn sắp xếp theo ngày để kết hợp với text box để tìm kiếm. Tiện ích JavaScript trong Sandbox hỗ trợ với tiện ích browser-side trong việc xây dựng và làm tương hợp với QueryParser.

* JavaScript query construction and validation

Trong những chương trước có thể nhận thấy rằng QueryPaser đưa ra trực tiếp cho end user sử dụng và có thể đưa đến sự hỗn độn. Nếu bạn cung cấp một trang web tìm kiếm , bạn có thể nghĩ đến việc sử dụng JavaScript query trong Sandbox . javascript Sandbox bao gồm các file ví dụ HTML giống như tìm kiếm nâng cao của Google với các lựa chọn khi search cho user sử dụng.Hỗ trợ tất cả các field HTML gồm text, hidden text , radio buttons, single và multiple selects. Với từng field HTML cần phải có tương ứng với tên đó được gắn thêm hậu tố *Modifier* .Modifier field có thể là một hidden field để ngăn cản user có thể điều khiển được nó . JavaScript file cho tùy biến với những đặc điểm giống như debug thông báo cho bạn biết những cái gì đang xảy ra. Rất dễ dàng để sử dụng JavaScript . JavaScript Sandbox đã được tăng lên rất nhiều. Có nhiều tiềm năng trong HTML , chúng ta có thể xem các ví dụ trong Sandbox khi cần biết sự hữu dụng của nó.



* Escaping special characters

QueryParser sử dụng nhiều ký tự đặc biệt , vì vậy những ký tự đó cần phải được tránh nếu như chúng sử dụng trong field name hoặc một phần của term. Sử dụng lucene- QueryEscaper.js hỗ trợ từ Sandbox , bạn có thể escape một query string.Bạn chỉ nên sử dụng query escape đối với fields và strings mà nó không chứa bất kỳ một ký tự đặc biệt nào.

* Using JavaScript support

Thêm JavaScript hỗ trợ file HTML có thể thêm đường dẫn file js vào trong thẻ <head> như 

Gọi hàm doMakeQuery để xây dựng một truy vấn và doCheckLuceneQuery để xác nhận câu truy vấn. Phương thức require một đối số của form field được định rõ nơi nào được đưa ra và thông qua. Escape query string sẽ bị gọi trở lại.

### Synonyms from WordNet

Có một hệ thống phát triển tại Princeton Univer-sity’s Cognitive Science Laboratory, driven by Psychology Professor George Miller minh họa hệ thống từ đồng nghĩa. WordNet được hình dung là dạng của từ ngữ có thể chuyển đổi cho nhau vả về từ vựng và ngữ nghĩa. Theo định nghĩa của Google thì đặc trưng thường nói đến những user trực tuyến hệ thống WordNet, đồng ý cho thông qua từ nối liền với nhau. Có thể xem hình ảnh sau cho hiện kết qyar tìm kiếm từ “search” tại WordNet.

Dữ liệu về các từ đồng nghĩa trong WordNet có thể được sử dụng trong Lucene Index. Nó đồng ý cho tìm kiếm nhanh những từ đồng nghĩa được đưa vào trong suốt quá trình indexing hoặc querying.

* Building the synonym index

Để cài đặt cho synonym index cần download và giải nén source code từ WordNet tại trang <http://www.cogsci.princeton.edu/~wn> , sau đó xây dựng synomym index sử dụng Syns2Index từ command line.Gồm 2 đối số : đối số 1 là đường dẫn đến file wn\_s.pl và đối số 2 là đường dẫn nơi Lucene index sẽ được tạo.

Sử dụng cơ sở dữ liệu từ WordNet để Lucene index field *word* và không index lại field đối với những từ đồng nghĩa (*syn*) cho từng dữ liệu đưa vào. Phiên bản 1.6 của WordNet có thể làm việc 39.718 documents và thực hiện với từng từ trong từng dữ liệu và cho ra file index với kích cỡ 2.5MB , với kích thước này nó có thể đủ để nạp vào RAMDirectory truy cập với tốc độ nhanh chóng.

Một tiện ích thứ 2 của chương trình WordNet Sandbox là giúp tìm kiếm những từ đồng nghĩa của một từ cho trước.

* Tying WordNet synonyms into an analyzer

Trong phần custom SynonymAnalyzer có thể dễ dàng kết hợp với WordNet sử dụng SynonymEngine . Có thể sử dụng với SynonyAnalyzer, điều chỉnh SynonymAnalyzerViewer sử dụng WordNetSynonymEngine .

Lucene WordNet có source code trên Lucene Sandbox tại địa chỉ <http://jwn.sourceforge.net/>

* Calling on Lucene

Lucene cpnf được dùng trong điện thoại , do yêu cầu trong điện thoại có kích thước bộ nhớ nhỏ nên kích thước của WordNet cũng được thu nhỏ lại cho phù hợp .T9 interface hiện tại được phổ dụng nhất trong phone.

Việc tìm kiếm trên màn hình điện thoại di động thì nhanh và đúng đắn thuyết phục . Index trong Lucene thì sử dụng hơn 2MB không phù hợp lắm với bộ nhớ của một số mobile-phone hiện tại .

### Highlighting query terms

Cho người sử dụng có thể tìm kiếm một vài nội dung xung quanh những thứ cần tìm và có liên quan với nó, thực sự điều đó thì rất hữu dụng.Với từng kết quả , có thêm những highlighting đối với những terms của truy vấn , nó sẽ tóm tắt được những nội dung bao quanh tìm kiếm những term truy câu truy vấn , và như thế cũng đủ để biết được và đánh giá sự đúng đắn của kết quả tìm kiếm .

Mark Harwood đã đóng góp vào Sandbox về highlight text . Ví dụ về highlighting với term truy vấn là *ipsum*.

Code về Highlight gần đây đã có những bước phát triển vững chắc trở thành một tiện ích tinh vi và linh động. Hightlighting gồm 3 phần chính: Fragmenter, Scorer, Formatter. Analyzer tạo ra TokenStreams, term trong Query thì được kết nối với Tokens đưa ra từ TokenStream. Những đoạn text nào giống với term ở trong đoạn text gốc thì được highlight .

Đi kèm với SimpleFragmenter chia cắt đoạn text gốc thành những đoạn fragments có kích thước giống nhau với mặc định là 100 ký tự. Highlight sẽ thực hiện lần lượt với từng đoạn.

QueryScore đi kèm trong Scorer .Công việc chính của Scorer sắp xếp thứ hạng của fragments. QueryScorer sử dụng terms từ truy vấn (Query) .

Formatter trang trí cho term text , làm nổi bật với những đoạn text trùng với dữ liệu được đưa vào truy vấn.

* Highlighting with CSS

Sử dụng tag <B> xung quanh đoạn text được highlight trước khi đưa ra cho người dùng xem hoặc có thể dùng CSS để thay thế . Ví dụ custom thẻ bắt đầu và kết thúc để highlight term dùng một thẻ <span> sử dụng CSS với class highlight. Sử dụng các thuộc tính của CSS , color , forrmating cho phần được highlight .

* Highlighting Hits

Những nơi chứa field text ban đầu trong index thì để cho bạn quyết định . Nếu đoạn text ban đầu không chứa trong index nó sẽ cho bạn quyết định lấy lại đoạn text được highlight từ gốc ban đầu. Nếu text được chứa trong field, nó có thể được lấy lại trực tiếp từ Document lấy từ Hits.

### Chaining filters

Sử dụng bộ lọc cho dữ liệu đầu vào để làm hẹp không gian tìm kiếm từ một truy vấn. Sandbox có chứa một bộ lọc đáng chú ý meta-filter được đóng góp bởi Kelvin Tan , nó có thể chuyển móc nối các bộ lọc lại với nhau và thi hành với AND , OR, ANDNOT có thể phối hợp với nhau. Dựa và những mục tiêu cần làm mà bạn có thể tạo ra những filter theo ý muốn để sử dụng.

### Sorting an index in Berkeley DB

Dự án không quá sôi nổi của Chandler (<http://www.osafoundation.org> ) được đưa ra để cố gắng xây dựng một mã nguồn mở Personal Information Manager. Chandle nhắm đến việc quản lý các loại thông tin như email, messages, appointments, contacts, tasks, notes, web pages, blogs, bookmarks, photo, .v.v. Điều bí mật phía bên trong của Chandler đó là sử dung Sleepycat’s Berkeley DB trong một không gian lớn có thể kết nối cơ sở dữ liệu thông qua nhiều con đường khác nhau. Chandler có code sử dụng ngôn ngữ Python.

# LUCENE PORTS

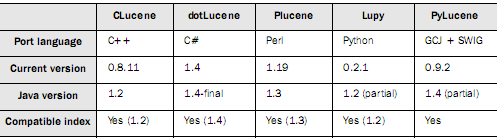
Các nội dung chính được đưa ra và giải quyết trong chương này:

* Sử dụng Lucene ports đến các ngôn ngữ khác nhau.
* So sánh các APIs của các ports , nét đặc trưng, và thực thi.

Trong những năm gần đây số người quan tâm và tham gia vào các dự án liên quan đến Lucene ngày càng đông . Lucene được đưa vào trong thực tế dưới dạng thư viện mã nguồn mở Java IR , Java hiện tại là một trong những ngôn ngữ phổ biến trong giới lập trình viên , tuy vậy nhưng không phải ai cũng biết sử dụng chuyên nghiệp Java. May thay , một số port của Lucene được tạo ra chấp nhận cho các ngôn ngữ khác , trong chương này sẽ nói vắn tắt một số port đã và đang được sử dụng .

### Ports’ relation to Lucene

Các port liên quan đến Lucene được thể hiện rõ qua bảng chi tiết sau:



Đặc trưng của mỗi port thì nó là một dự án độc lập với nhau. Với từng port đều có cung cấp hỗ trợ cho những người quan tâm về nó rất kỹ lưỡng ví dụ website , link download port , địa chỉ mail liên hệ , và nhiều thứ khác . Mỗi port còn có giới thiệu nhóm tác giả và những lập trình viên đang phát triển nó.

Hầu hết các port đều đang cố gắng đồng bộ hóa mã nguồn với những phiên bản mới nhất của Lucene.

### Clucene

Clucene là mã nguồn mở dùng để phát triển cho port của Apache Lucene từ ngôn ngữ C++. Nó được phát hành với giấy phép LGPL và địa chỉ quan tâm <http://sourceforge.net/projects/clucene> . Tại thời điểm viết sách phiên bản hiện tại là 0.8.11 dựa trên phiên bản 1.2 của Lucene. Tới thời điểm hiện tại thì đã cho ra phiên bản 0.9.21b , ngày cập nhật mới nhất là ngày 23/10/2008 . Các gói của CLucene được tích hợp vào cũng giống như trong phiên bản Lucene , gồm cả các gói để chấp nhận cho CLucene được sử dụng với các ngôn ngữ khác. Hiện tại có các gói cho PHP, .Net , Dynamic Link Library(DLL) được chia sẻ giữa các ngôn ngữ với nhau.

* Supported platforms: CLucene được khởi tạo từ Microsoft Visual Studio nhưng giờ nó còn được compiles trong GCC, MinGW32, Borland C++ compiles. Ngoài hệ điều hành Windows , CLucene còn được build thành công trong hệ điều hành Red Hat 9, Mac OS X . Các nhà phát triển còn đang thực hiện hỗ trợ thêm với kiến trúc ADM64, FreeBSD.
* API compatibility: CLucene API tương tự như Lucene .Vấn đề còn đang được giải quyết đó là những ký tự ngoài dãy ASCII .
* Unicode support : được viết dựa trên nền Linux cũ chưa hỗ trợ cho Unicode , vì vậy CLucene không hỗ trợ tốt với các ký tự Unicode . Trong tương lai gần các phiên bản mới có thể giải quyết vấn đề tùy ý lựa chọn thêm vào các thư viện Unicode.
* Users: trên thực tế các nhóm phát triển CLucene thì nhỏ và các bước thực hiện dự án gặp nhiều khó khăn đặc trưng khi thực hiện. Awasu là một người quản lý có kiến thức về các công cụ sử dụng CLucene (http://www.awasu.com/).

### DotLucene

Lucene.Net được phát hành với giấy phép Apache Software License (ASL) , source code có thể tìm thấy từ link <http://www.sourceforge.net/projects/dotlucene/> , sự phân bổ các gói của dotLucene giống như Lucene. Bao gồm source code , test , một vài ví dụ demo.

Ngoài port dotLucene còn có một port khác của Lucene trên nền tảng .Net là Nlucene , link <http://www.sourceforge.net/projects/nlucene/> phiên bản mới nhất đã là từ năm 2002 và nó không chuyển đổi đầy đủ.

* API compatibility: Dù được viết bằng C# nhưng nó xuất ra mọt API gần giống y hệt Lucene.Vì vậy code được viết cho Lucene có thể được port qua C# với sự tác động nhỏ.Sự khác biệt đó là giới hạn của Java và C# về phong cách đặt tên. Những nơi tên của phương thức trong Java thì viết bằng ký tự viết thường, còn bên .Net thì nó được viết in hoa.
* Index compatibility: dotLucene thì tương đương với Lucene trong index level , một index được tạo ra bởi Lucene có thể đọc được bởi dotLucene và ngược lại.Tuy nhiên do Lucene thường xuyên được cập nhật phiên bản mới cho nên nó chỉ tương tích với nhau trong giới hạn từ phiên bản 1.4 , phiên bản hiện tại của Lucene là 2.4.1 (9/3/2009).
* Users: những user nổi tiếng nhất được tìm thấy trong link

<http://www.lookoutsoft.com/Lookout/> , ví dụ như một người thích thú với Lucene.Net như Beagle <http://www.gnome.org/projects/beagle/> đã xây dựng một GNOME componet cho indexing và searching cho tất cả các loại file bao gồm cả hình ảnh.

### Plucene

Plucene là một port của Lucene được viết trên ngôn ngữ Perl, có thể tìm thấy trong CPAN <http://search.cpan.org/dist/Plucene/> . Phiên bản 1.19 được phát hành năm 2004 , vì vậy nó có sự mới lạ với phiên bản 1.3 của Lucene. Hầu hết công việc đã được làm bởi Simon Cozens.

* API compatibility: Nó là port trực tiếp của Lucene , Plucene lĩnh hội được phần lớn các hàm API , và hiển nhiên sẽ có sự khác nhau về cách thức đặt tên trong code , cấu trúc , modules, classes, method.
* Index compatibility: Theo tác giả của Plucene , index được tạo bởi Lucene 1.3 , và Plucene 1.19 thì tương hợp , có thể đọc được từ nội dung được index bởi lẫn nhau.
* Performance: phiên bản 1.19 thì chậm một cách đáng kể so với Lucene ở phiên bản Java.
* Users: theo một cố vấn của Plucene , Plucene thì được sử dụng bởi Gizmodo (<http://www.gizmodo.com/> ), một site xem xét về các thiết bị điện tử cho khách hàng , hay còn được Twingle sử dụng <http://www.twingle.com> .

### Lupy

Lupy là một port của Lucene 1.2 từ ngôn ngữ Python .Những nhà phát triển chính của Lupy là Amir Bakhtiar và Allen Short . Một vài hàm trong Lucene thì bị lỗi trong Lupy như QueryParser , một vài phân tích , index mergin , locking, và một vài items nhỏ . Dù Lupy là một port của một phiên bản Lucene cũ nhưng nó đang dần hoàn thiện để đưa ra một phiên bản mới gần nhất với phiên bản 1.4 . Phiên bản hiện tại của Lupy là 0.2.1, có thể tìm thấy trên trang <http://www.divmod.org/Home/Projects/Lupy/>.

* API compatibility: Lupy API thì tương đồng với Lucene, có sự giống nhau về các classes, methods. Tuy nhiên chúng ta có thể tạo một IndexWriter với chỉ định một analyzer làm một vài thứ mà chúng ta không thể làm trong Lucene.
* Index compatibility: một index được tạo với Lupy thì tương thích với Lucene. Tính tương hợp thì bị giới hạn bởi một phần của phiên bản , index thì chỉ tương đồng với Lucene phiên bản 1.2 .
* Performance: Giống như Plucene , Lupy là một port trực tiếp từ Lucene gốc, chưa có bí quyết nào ghi rõ trong Lupy thì chắc chắn tối ưu trong thực thi hơn Python port.
* Users: người sử dụng chính của Lupy là Divmod <http://www.divmod.com/> .

### PyLucene

PyLucene là một port gần đây nhất của Lucene. Nó được đăng ký bởi MIT license, và người đứng tên là Andi Vajda người có công đóng góp Berkeley DbDirectory trong code chuẩn của Lucene. Nó được indexing và searching component của Chadler , và có thể mở rộng trong cộng đồng mã nguồn mở PIM , nhưng nó trở thành một project riêng biệt trong năm 2004 . Bạn có thể tìm thấy PyLucene tại link <http://pylucene.osafoundation.org/> .

* API compatibility: Không phải tất cả các hàm của Lucene thì dùng được trong PyLucene. Từ khi PyLucene trở thành một project độc lập , người sử dụng có nhiều yêu cầu hơn từ nó, Adi và nhóm phát triển từ từ trình bày nhiều hơn về Lucene API thông qua SWIG . Các cấu trúc liên quan các hàm API thì hầu như giống với Lucene , dễ dàng cho người sử dụng Lucene học cách sử dụng PyLucene .
* Index compatibility: Vì sự tự nhiên của PyLucene (“compiler and SWIG gymnastics”) , index của nó tương thích với Lucene.
* Performance: Mục đích của PyLucene không thể là port nhanh nhất của Lucene nhưng có thể là port gần giống nhất.
* Users: do nó là port mới của Lucene nên chưa có một public user , tuy nhiên có một project được biết đến được xây dựng bởi Chandle <http://www.osafoundation.org/> .

### TÀI LIỆU THAM KHẢO

#### [.] Lucene In Action . A guide to the Java search engine . Otis Gospondetic Erick Hatcher – Foreword by Doug Cutting.