SEPT- SEARCH ENGINE AND PROCESSING TOOL (QUERY DSL)

Adrian FILIP, Vadim DOGA, Tatiana POLEACOV, Nina CAVCALIUC

Technical University of Moldova

Abstract: *The Query DSL is distinct from other query languages, it has to work with text files and execute the search using a predefined pattern or expression via some efficient function. Their main purpose is to be useful to use by non-programmers within their field of activity. In this article, the grammar construction using the ANTLR is described. Also, there are described predefined functions which will be implemented in the process of development of this DSL, shown to be efficient and easy to use.*

Keywords: *DSL, language, lexicon, grammar.*

Introduction

Nowadays, there are a lot of problems in text processing [1] area that are divided into two classes: problems that are similar to those of business, and those that are unique to information science. The former consists of front-office applications, such as word processing, spreadsheets and desktop computing, and back-office applications, comprising data processing functions like accounts payable and receivable, payroll and other business operations.

These issues derive from the need of text manipulation — the ability to analyze, process and reformat the text. Information retrieval is one of the beneficiary areas. The aim is to rapidly select relevant information by applying Boolean logic to keywords and by searching databases which are optimized for textual storage and retrieval. The next area is linguistic research and natural language processing. Textual analysis answers all manner of research questions. The most used area is word processing. This one is based on typists who check completed work for spelling, grammar, punctuation or type correspondence, reports, text and other written material from rough drafts.

Text Query DSL suits especially the typists that work intensive with text files. It will be a great tool for those that are always in need of looking after their spelling and the usage of the words, mainly in order to eliminate the word repetition. This Text Query tool can search through files and check in which lines from the files had matched a given regular expression matches. Searching and replacing with regular expressions is a powerful way to maintain all sorts of text files. It will output the line numbers or the actual lines as string that matched the specified expression. This DSL will also print all the occurrences of the given pattern and will output this number on the screen. It computes and prints the total amount of the words from the input file, which is advantageous when the person needs to work with a limited number of words.

1. Lexical Analysis

In order to build and develop a domain specific language, one has to analyze how the features are going to look like based on the functionalities it is supposed to provide. Nevertheless, this implies the keywords, in case it has a start and an end of the program and all the rules which give the possibility to understand and write clean code.

All SPET keywords are lowercase except two words that define the start and the end of the program, which means that it is a case-sensitive language. For example, **Start** is a keyword that highlights the start of the program, but *start* is a word that the program will find it in a text, also this means that **Start** cannot be looked for as a search word.

The reserved words are: **Start, End, lookFor, find, findL, findR, count, replace.** White spaces may appear only as a new line between keywords, it is defined as one or more spaces, tabs and line-breaking. Only after the keyword **Start** and **End** there can be white space. On the other hand, for other identifiers, space between them will be interpreted as one string, for example *oldman* and *old man* are one string, but not two for the last.

For defining the lexicon of DSL, its grammar should be made in a general form, which may be done by identification of all terminal and nonterminal characters. In order to classify all of them, they should be divided into two sets, after which the derivation from nonterminal into terminal will be done.

<program >	Means program is a nonterminal.
Start	(in **bold** font) means that **Start** is a terminal / a token or a part of token
X^*	Means zero or more occurrences of *X*.
X^+ ,	A comma-separated list of one or more x's.
|	Separate alternatives.

Fig.1 "Meta-notation".

Below is presented the grammar of the DSL:

V_N = { <program> , <func_call> , <type>, <funct_name> , <exp> , < alpha> , <string_literal>,<filePath>, <digit>, <character>, <symbol>}

V_T = { Start ,End , find ,findL ,count , findR , replace, ; , " ,' , (,) , = , Aa..zZ , 1..9, - , / , [,] , * ,^ , < , > }

<program> → **Start** <func_call>* **End**

<func_call> → <func_name> (<filePath> , [<exp> $^+$,]) ;

<funct_name> → **lookFor|find | findL |findR | replace | count**

<exp> → <alpha> | <string_literal> | <symbol>

<string_literal> → **"** <alpha> **"**

<alpha> → < digit > | < character > |

<filePath> → **(**<character>**|** <digit> **|':'|'\\'|'/'|' '|'-'|'_'|'.')**$^+$

<digit> → **| 0|..|9||**

<character> → **a|b|..|z|A|..|Z|**

<symbol> → **- | / | [|] | * | ^ | < | > |**

A SEPT program consists of a single main method which begins with Start and finishes with End. The entire program will be analyzed from the top – **Start** to **End**, these keywords must be written from a new line in order to process and interpret it faster. Between them, predefined functions will be used, which also must be instantiated from a new line and finish the statement using a semicolon, but the declaration of multiple functions into one line is also allowed. This will not invoke any errors.

A SEPT program consists of a single main method which begins with Start and finishes with End. The entire program will be analyzed from the top – **Start** to **End**, these keywords must be written from a new line in order to process and interpret it faster. Between them, predefined functions will be used, which also must be instantiated from a new line and finish the statement using a semicolon, but the declaration of multiple functions into one line is also allowed. This will not invoke any errors.

Predefined functions:

find– this function searches for a specified whole word or a single character in a text. This search is case sensitive, it means that a word which starts with capital letter is different from the same word but written with a non-capital character. It accepts two parameters: first one is the file path which will be processed, while the second one is what the user is looking for and returns where that lexeme occurs: line and position in that line.

Syntax: find(file path, word/character);

Example: lookFor(C:\Users\Desktop\grammar.txt, man);

findL– will return a single line as a string. As parameters it gets: path of the file, number of line.

Syntax: findL(file path, number of line);

Example: findL(C:\Users\Desktop\grammar.txt, 5);

findR – returns the number of character's repetition. Syntax is the same as of **find** function**.**

lookFor – is a function based on regex [2], because of which it will search using some constraints, highlighted by special symbols, such as: ^, ?, < >,/, *, [ ]. Unlike the **find** function, the range of utilities is bigger, it does not look only for a whole word but also for certain characters, therefore it will retrieve the word that has that piece of string and where it occurs.

Just like the **find** function, **lookFor** accepts the same parameters, although the second one must include special symbols. Each one has their meaning: if using caret sign ^ , it will seek all words that start or end with that letters depending on their location. If we look for prefix, this sign will be positioned before the word, while for suffix it will occur after.

The question sign is used between two letters and it finds the word that starts and ends with them. “Greater than” and “less than” signs are used together < character string >, and will retrieve all terms that contain that string no matter where it is localized. Asterisk is used to find re-occurrences of the previous character, for example *lo*t* will find *lot* or *loot*.

In order to look for any digit, character or sequence they have to be put between two forward slashes, also if we want to find a certain number of consecutive appearances we can indicate it after the last slash, for example 2/0-9/ will search the text entity where two digits appear one after another: 01 or 95. This function has a special feature such as finding the sequences of words or characters respecting a format, for which square brackets [ ] are used, between which the exact format is indicated e.g [dd'/'dd'/'dddd] where ‘d’ means digit in this case, we look for a date/time format and it can return 12/01/2012. To avoid special symbols interpretation these will be used between single quotation marks.[3]

Syntax: lookFor(file path, word/character with special symbols);

replace – replaces in the text one character with another or a word with another and does not return anything. It has three parameters: the first two are the same as of **find,** only the third is the word/character that must be inserted instead of the first one.

Syntax: replace(C:\Users\Desktop\grammar.txt, word, new word);

count – returns the total number of words from a file. In comparison with others it has only one parameter - the file path.

Syntax: count(C:\Users\Desktop\grammar.txt);

Conclusions

This paper is meant to show the use of a Text Query DSL, which will be designed in a way that will ease the process of working with text files. Using the ANTLR, the grammar for this domain specific language has been designed, which helped with the function construction. Whether it has to be an article or an essay with a limited number of words, or an important document needed for a meeting, it is a very useful tool that will help the person monitor their written texts. The SEPT language will let the owner of the file easily navigate through the text file and make manipulations where needed. Taking into considerations the points mentioned above, it is a great time-saving tool and really easy to handle, which is an advantage for those who are non-programmers.

References

1. https://medium.com/krakensystems-blog/text-processing-problems-with-non-english-languages-82822d0945dd
2. https://en.wikipedia.org/wiki/Regular_expression
3. http://www.gmayor.com/replace_using_wildcards.htm
4. Markus Völter *DSL Engineering: Designing, Implementing and Using Domain-Specific Languages, 2013*