Auto-Complete Suggestion Basing on Solr

May, 2018

515030910553 Sun Rui

Abstract

Auto-complete is a useful application to predict the word a user wants to type, and in the search engine it is known as auto-suggest. This project is meant to realize the suggester function basing on the apache solr search engine.The suggester completion may come from a dictionary that is based upon the main index or upon any other arbitrary dictionary, data or file. The result will be only top-N suggestions, either ranked alphabetically or according to their usefulness for an average user or weight defined by the system.

1 Backgrounds

1.1 Auto-Complete Search

Auto-complete, or word completion, is a feature in which an application predicts the rest of a word a user is typing. In graphical user interfaces, users can typically press the tab key to accept a suggestion or the down arrow key to accept one of several.

Auto-complete speeds up human-computer interactions when it correctly predicts the word a user intends to enter after only a few characters have been typed into a text input field. It works best in domains with a limited number of possible words (such as in command line interpreters), when some words are much more common (such as when addressing an e-mail), or writing structured and predictable text (as in source code editors).

In search engines, auto-complete user interface features provide users with suggested queries or results as they type their query in the search box. This is also commonly called auto-suggest or incremental search. This type of search often relies on matching algorithms

that forgive entry errors such as phonetic Soundex algorithms or the language independent Levenshtein algorithm. The challenge remains to search large indices or popular query lists in under a few milliseconds so that the user sees results pop up while typing.

1.2 Solr Search Engine

Solr (pronounced "solar") is an open source enterprise search platform, written in Java, from the Apache Lucene project. Its major features include full-text search, hit highlighting, faceted search, real-time indexing, dynamic clustering, database integration, NoSQL features and rich document (e.g., Word, PDF) handling. Providing distributed search and index replication, Solr is designed for scalability and fault tolerance. Solr is widely used for enterprise search and analytics use cases and has an active development community and regular releases.

Solr runs as a standalone full-text search server. It uses the Lucene Java search library at its core for full-text indexing and search, and has REST-like HTTP/XML and JSON APIs that make it usable from most popular programming languages. Solr's external configuration allows it to be tailored to many types of application without Java coding, and it has a plugin architecture to support more advanced customization.

Apache Solr is a powerful tool with tremendous search capability. In order to search a document, it performs following operations in sequence:

1. Indexing: First of all, it converts the documents into a machine-readable format which is called Indexing.

2. Querying: Understanding the terms of a query asked by the user. These terms can be images, keywords, and much more.

3. Mapping: Solr maps the user query to the documents stored in the database to find the appropriate result.

4. Ranking the outcome: As soon as the engine searches the indexed documents, it ranks the outputs as per their relevance.

2 Auto complete suggestion on Solr

A common need in search applications is suggesting query terms or phrases based on incomplete user input. These completions may come from a dictionary that is based upon the main index or upon any other arbitrary dictionary. It's often useful to be able to provide only top-N suggestions, either ranked alphabetically or according to their usefulness for an average user (e.g. popularity, the number of returned results, or the weighted defined by the user).

The Suggester Component in Solr provides users with automatic suggestions for query terms.This approach utilizes Lucene's Suggester implementation and supports all of the lookup implementations available in Lucene.

The main features of this Suggester are:

1. Lookup implementation pluggability
2. Term dictionary pluggability, giving you the flexibility to choose the dictionary implementation
3. Distributed support

In my solr auto-complete project, I realize this function basing the spell check part and define the suggest component and request handler. The search can be based on database or file-based data.

3 Suggest Component

In the suggest component, we reuses much of the SpellCheckComponent infrastructure.

The code are as below:

```
<searchComponent class="solr.SpellCheckComponent" name="
    suggest" >
 <str name="queryAnalyzerFieldType">string</str>
 <lst name="spellchecker">
    <str name="name">default</str>
<str name="field">_entext_</str>
    <str name="classname">org.apache.solr.spelling.suggest.
        Suggester</str>
    <str name="lookupImpl">org.apache.solr.spelling.suggest.
        tst.WFSTLookupFactory</str>
    <float name="threshold">0.005</float>
<str name="sourceLocation">tmp.txt</str>
<str name="spellcheckIndexDir">dic</str>
<str name="comparatorClass">freq</str>
    <str name="buildOnOptimize">true</str>
    <str name="buildOnCommit">true</str>
 </lst>
</searchComponent>
```

3.1 Suggestion on dadtabase

In this part, the main different parameters define as below:

1. queryAnalyzerFieldType: The fieldType type in managed-schema. If this option is added, the spelling checker will call this fieldType tokenizer. If it is not added, Solr will take the word splitter of field defined below or just created a segmentation based

on space because SpellCheckComponent requires a word breaker to run. In our project, we now hope that Analyzer does not make any changes to the query, so select string.

2. name: Same as the field defined in request handler.

3. lookupImpl: The look-up of matching suggestions in a dictionary is implemented by subclasses of the Lookup class. There are four main implementations that are included in solr:

   - JaspellLookup - tree-based representation based on Jaspell
   - TSTLookup - ternary tree based representation, capable of immediate data structure updates
   - FSTLookup - finite state automaton based representation
   - WFSTLookup - weighted automaton representation: an alternative to FSTLookup for more fine-grained ranking

   For practical purposes all of the above implementations will most likely run at similar speed when requests are made via the HTTP stack. Direct benchmarks of these classes indicate that (W)FSTLookup provides better performance and has a much lower memory cost compared to the other two methods.(JaspellLookup can provide "fuzzy" suggestions, but this functionality is not currently exposed).

4. threshold: A value between 0 and 1, representing the minimum fraction of documents where a term should appear. This will limiting some infrequent words. But if the search is a file-based dictionary, this parameter will be useless.

5. comparatorClass: Return the sort of the result. There are four different types:

   - empty: If not settled, the default will call this
   - score: Explicitly choose the default case

- freq: Sort by frequency first, then score
- a fully qualified class name: Provide a custom comparator that implements Comparator

6. buildOnOptimize/buildOnCommit: If set to true then the Lookup data structure will be rebuilt after commit/optimize. If false (default) then the Lookup data will be built only when requested. Noticing that only when this parameter is set as true the spellchecker will work.

3.2 Suggestion on dictionary

When a file-based dictionary is used then it's expected to be a plain text file in UTF-8 encoding. Each line must be consist of either a string without literal TAB character, or a string and a TAB separated floating-point weight.

If weight is missing, it will be assumed as 1.0. Weights affect the sorting of matching suggestions when $spellcheck.onlyMorePopular = true$ is selected - weights are treated as "popularity" score.

This part is defined in the suggest component as:

```
1    <str name="sourceLocation">tmp.txt</str>
2    <str name="spellcheckIndexDir">dic</str>
```

In this section, spellcheckIndexDir is the index file directory for recommended check. Solr will create this folder at startup and store the index of the check suggestion under this folder.

4 Request Handler

After defining the suggest component, we need to add a new handler that uses SearchHandler with with SearchComponent that we just defined.

The code:

```
<requestHandler name="/suggest" class="solr.SearchHandler">
    <lst name="defaults">
      <str name="spellcheck.dictionary">default</str>
      <str name="spellcheck">true</str>
   <str name="spellcheck.onlyMorePopular">true</str>
      <str name="spellcheck.count">10</str>
      <str name="spellcheck.collate">true</str>
   <!--str name="spellcheck.build">true</str-->
    </lst>
    <arr name="components">
      <str>suggest</str>
    </arr>
 </requestHandler>
```

And the default component are defined as below:

1. spellcheck: Run the Suggester for queries submitted to this handler.
2. count: Configure the number of spell check prompt results.
3. onlyMorePopular: If this parameter is set to true then the suggestions will be sorted by weight ("popularity") - the count parameter will effectively limit this to a top-N list of best suggestions. If this is set to false then suggestions are sorted alphabetically.
4. collate: Provide a query collated with the first matching suggestion.
5. build: Use this function to reconstruct the index when the core reloads.

5 Result

5.1 Data search result

If we put "ac" as the input qt basing on , the result is:


```
{
  "responseHeader":{
    "status":0,
    "QTime":14},
  "spellcheck":{
    "suggestions":[
      "ac",{
        "numFound":5,
        "startOffset":0,
        "endOffset":2,
        "suggestion":["activ",
          "academi",
          "acid",
          "acta",
          "acut"]}],
    "collations":[
      "collation","activ"]}}
```


5.2 Dictionary search Result

The give dictionary type is as(paper_name paper_ID TAB weight):

```
illinois college of optometry 000011FD 409
sangji university 00002523 820
manchester institute of innovation research 00003DEF 101
ateneo de manila university 00004D0A 1010
instituto militar de engenharia 0000A38E 1094
```

```
6 sapientia university 0000AB6E 123
7 kahramanmaras sutcu imam university 0000B3C1 1646
8 al akhawayn university 0000BAE4 364
```

If we put the "ac" as the input qt, the result is:


```
{
  "responseHeader":{
    "status":0,
    "QTime":32},
  "spellcheck":{
    "suggestions":[
      "ac",{
        "numFound":10,
        "startOffset":0,
        "endOffset":2,
        "suggestion":["academia sinica 050BB43F",
          "academy of sciences of the czech republic 0C103FFF",
          "academy of medical sciences united kingdom 4DA9DEC2",
          "academy of engineering 027EBF4F",
          "academy of sciences of moldova 06AE5ED6",
          "acadia university 05864F21",
          "academia nacional de medicina 00DC2C5D",
          "academy of athens 0C494083",
          "acharya nagarjuna university 0A8CD272",
          "academy for urban school leadership 4DF0D510"]}],
    "collations":[
      "collation","(academia sinica 050BB43F)"]}}
```


It will return top ten weighted suggestions with name and the paper ID.

References

[1] https://en.wikipedia.org/wiki/Autocomplete

[2] https://en.wikipedia.org/wiki/Apache_Solr

[3] https://wiki.apache.org/solr/Suggester

[4] https://lucene.apache.org/solr/guide/6_6/solrcloud.html

[5] http://iamyida.iteye.com/blog/2205114

[6] http://public.dhe.ibm.com/software/dw/java/j-solr1-pdf.pdf