SpellCheckComponent

<!> Solr1.3

Introduction

The SpellCheckComponent is designed to provide inline spell checking of queries without having to issue separate requests. Another and possibly clearer way of stating this is that it makes query suggestions (as do well-known web search engines), for example if it thinks the input query might have been misspelled. (Some people tend to think that "spellchecker" is actually a misnomer, and something along the lines of "query suggest" would have been more appropriate.)

The SpellCheckComponent can use the Lucene SpellChecker to give suggestion for given words, or one can implement their own spell checker in Java using the SolrSpellChecker abstract base class.

Quick Start

The example server is configured with a request handler that uses the SpellCheckComponent. If you haven't already, start the example server and index the example data as shown in the solr tutorial.

Now send a spellcheck request. Note the **spellcheck.build=true** which is needed only once to build the spellcheck index from the main Solr index. It takes time and should **not** be specified with each request.

```
http://localhost:8983/solr/spell?q=delll ultrashar&spellcheck=true&spellcheck.collate=true&spellcheck.build=true
```

The response contains words not found in the index, along with alternatives. The **spellcheck.collate=true** causes a modified version of the original query to be returned with the most likely alternatives.

```
<lst name="spellcheck">
  <lst name="suggestions">
    <lst name="delll">
      <int name="numFound">1</int>
      <int name="startOffset">0</int>
      <int name="endOffset">5</int>
      <arr name="suggestion">
        <str>dell</str>
      </arr>
    </lst>
    <lst name="ultrashar">
      <int name="numFound">1</int>
      <int name="startOffset">6</int>
      <int name="endOffset">15</int>
      <arr name="suggestion">
        <str>ultrasharp</str>
      </arr>
    </lst>
    <str name="collation">dell ultrasharp</str>
  </lst>
</lst>
```

Configuration

The first step to configure the SpellCheckComponent is to specify the source of words which should be used for suggestions in solrconfig.xml. The words can be loaded from a field in Solr, text files or even from fields in arbitary Lucene indices. A sample configuration for loading words from a field in Solr looks like the following:

```
<searchComponent name="spellcheck" class="solr.SpellCheckComponent">

  <lst name="spellchecker">
    <!--
        Optional, it is required when more than one spellchecker is configured.
        Select non-default name with spellcheck.dictionary in request handler.
    -->
    <str name="name">default</str>
    <!-- The classname is optional, defaults to IndexBasedSpellChecker -->
    <str name="classname">solr.IndexBasedSpellChecker</str>
    <!--
        Load tokens from the following field for spell checking,
        analyzer for the field's type as defined in schema.xml are used
    -->
    <str name="field">spell</str>
    <!-- Optional, by default use in-memory index (RAMDirectory) -->
    <str name="spellcheckIndexDir">./spellchecker</str>
    <!-- Set the accuracy (float) to be used for the suggestions. Default is 0.5 -->
    <str name="accuracy">0.7</str>
    <!-- Require terms to occur in 1/100th of 1% of documents in order to be included in the dictionary -->
    <float name="thresholdTokenFrequency">.0001</float>
  </lst>
  <!-- a spellchecker that can break or combine words. (Solr 4.0 see SOLR-2993) -->
  <lst name="spellchecker">
    <str name="name">wordbreak</str>
    <str name="classname">solr.WordBreakSolrSpellChecker</str>
    <str name="field">spell</str>
    <str name="combineWords">true</str>
    <str name="breakWords">true</str>
    <int name="maxChanges">3</int>
  </lst>
  <!-- Example of using different distance measure -->
  <lst name="spellchecker">
    <str name="name">jarowinkler</str>
    <str name="field">lowerfilt</str>
    <!-- Use a different Distance Measure -->
    <str name="distanceMeasure">org.apache.lucene.search.spell.JaroWinklerDistance</str>
    <str name="spellcheckIndexDir">./spellchecker</str>
```

```
  </lst>

  <!-- This field type's analyzer is used by the QueryConverter to tokenize the value for "q" parameter -->
  <str name="queryAnalyzerFieldType">textSpell</str>
</searchComponent>
<!--
    The SpellingQueryConverter to convert raw (CommonParams.Q) queries into tokens.  Uses a simple regular
expression
    to strip off field markup, boosts, ranges, etc. but it is not guaranteed to match an exact parse from the
query parser.

Optional, defaults to solr.SpellingQueryConverter
-->
<queryConverter name="queryConverter" class="solr.SpellingQueryConverter"/>

<!--  Add to a RequestHandler
     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
     NOTE:  YOU LIKELY DO NOT WANT A SEPARATE REQUEST HANDLER FOR THIS COMPONENT.  THIS IS DONE HERE SOLELY FOR
     THE SIMPLICITY OF THE EXAMPLE.  YOU WILL LIKELY WANT TO BIND THE COMPONENT TO THE /select STANDARD REQUEST
HANDLER.
     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
-->
<requestHandler name="/spellCheckCompRH" class="solr.SearchHandler">
  <lst name="defaults">
    <!-- Optional, must match spell checker's name as defined above, defaults to "default" -->
    <str name="spellcheck.dictionary">default</str>
    <!-- Also generate Word Break Suggestions (Solr 4.0 see SOLR-2993) -->
    <str name="spellcheck.dictionary">wordbreak</str>
    <!-- omp = Only More Popular -->
    <str name="spellcheck.onlyMorePopular">false</str>
    <!-- exr = Extended Results -->
    <str name="spellcheck.extendedResults">false</str>
    <!--  The number of suggestions to return -->
    <str name="spellcheck.count">10</str>
  </lst>
  <!--  Add to a RequestHandler
       !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
       REPEAT NOTE:  YOU LIKELY DO NOT WANT A SEPARATE REQUEST HANDLER FOR THIS COMPONENT.  THIS IS DONE HERE
SOLELY FOR
       THE SIMPLICITY OF THE EXAMPLE.  YOU WILL LIKELY WANT TO BIND THE COMPONENT TO THE /select STANDARD
REQUEST HANDLER.
       !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  -->
  <arr name="last-components">
    <str>spellcheck</str>
  </arr>
</requestHandler>
```

When adding `<str name="field">FieldName</str>` be aware all fieldType processing is done prior to the dictionary creation. It is best to avoid a heavily processed field (ie synonyms and stemming) to get more accurate results. If the field has many word variations from processing then the dictionary will be created with those in addition to more valid spell checking data.

Multiple "spellchecker" instances can be configured in the same way. The currently available spellchecker implementations are:

- org.apache.solr.spelling.IndexBasedSpellChecker – Create and use a spelling dictionary that is based on the Solr index or an existing Lucene index
- org.apache.solr.spelling.FileBasedSpellChecker – Create and use a spelling dictionary based off a flat file. This can be useful for using Solr as a spelling server or in other instances when spelling suggestions do not need to be based on the content of an actual index.
- org.apache.solr.spelling.DirectSolrSpellChecker ⚠️ LUCENE-2507] – Experimental spellchecker that only uses your main Solr index directly (build /rebuild is a no-op). See [https://issues.apache.org/jira/browse/LUCENE-2507 for more information.
- org.apache.solr.spelling.WordBreakSolrSpellChecker ⚠️ SOLR-2993] – Generates suggestions by Combining adjacent words and/or breaking words into multiples. This spellchecker can be configured with a traditional checker (ie: DirectSolrSpellChecker). The results are combined and collations can contain a mix of corrections from both spellcheckers. See [https://issues.apache.org/jira/browse/SOLR-2993 for more information.

Custom Comparators and the Lucene Spell Checkers (IndexBasedSpellChecker, FileBasedSpellChecker, DirectSolrSpellChecker)

⚠️ Solr3.1 Solr4.0

By default, the Lucene Spell checkers sort suggestions first by the score from the string distance calculation and second by the frequency (if available) of the suggestion in the index. Applications may wish to change this to better fit their scoring needs. This can be done when configuring the Lucene spell checker by adding the comparatorClass option to the configuration. This is a string value which may be one of the following:

- Empty – in which case the default is used.
- 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<SuggestWord>.

See SuggestWordScoreComparator in the contrib/spellchecker code base of Lucene for an example.

An example configuration might look like:

```
<lst name="spellchecker">
  <str name="name">freq</str>
  <str name="field">lowerfilt</str>
  <str name="spellcheckIndexDir">spellcheckerFreq</str>
  <!-- comparatorClass be one of:
       1. score (default)
       2. freq (Frequency first, then score)
       3. A fully qualified class name
  -->
  <str name="comparatorClass">freq</str>
  <str name="buildOnCommit">true</str>
</lst>
```

Building on Commits

SpellCheckComponent can be configured to automatically (re)build indices based on fields in Solr index when a commit is done. In order to do so you must enable this feature by adding the following line in your SpellCheckComponent configuration for each spellchecker where you wish it to apply:

```
<str name="buildOnCommit">true</str>
```

For example:

```
<lst name="spellchecker">
  <str name="name">default</str>
  <str name="field">spell</str>
  <str name="spellcheckIndexDir">./spellchecker1</str>
  <str name="buildOnCommit">true</str>
</lst>
```

⚠️ **NOTE**: Building on commit is very expensive and is discouraged for most production systems. For large indexes, one commit may take minutes since the building of spellcheck dictionary is single threaded. Use buildOnOptimize or explicit build instead.

Building on Optimize

⚠️ Solr1.4

SpellCheckComponent can be configured to automatically (re)build indices based on fields in Solr index when an optimize command is done. In order to do so you must enable this feature by adding the following line in your SpellCheckComponent configuration

```
<str name="buildOnOptimize">true</str>
```

thresholdTokenFrequency

For use with IndexBasedSpellChecker or DirectSolrSpellChecker. This specifies the percentage of documents in which a term must occur in order to be included in any spelling suggestions. (In the case of IndexBasedSpellChecker, only terms that meet this requirement will be indexed in the spelling dictionary.) For example, the following configuration line limits the dictionary to terms that occur in at least 1% of the documents:

```
<float name="thresholdTokenFrequency">.01</float>
```

Note that this does not affect whether or not a user's query is considered to be correctly spelled as these spell checkers never offer suggestions for terms included in the full original documents. However, specifying thresholdTokenFrequency will prevent low-instance terms from being offered as spelling suggestions.

Spell Checking Analysis

SpellCheckingAnalysis - Provides details on how Analysis and Spell Checking work together

Request Parameters

spellcheck

Turn on or off spellcheck suggestions for this request. If true, then spelling suggestions will be generated.

q OR spellcheck.q

The query to spellcheck. If spellcheck.q is defined, then it is used, otherwise the original input query is used. The spellcheck.q parameter is intended to be the original query, minus any extra markup like field names, boosts, etc. If the q parameter is specified, then the SpellingQueryConverter class is used to parse it into tokens, otherwise the WhitesepaceTokenizer is used. The choice of which one to use is up to the application. Essentially, if you have a spelling "ready" version in your application, then it is probably better to send spellcheck.q, otherwise, if you just want Solr to do the job, use the q parameter

Note: The SpellingQueryConverter class does not deal properly with non-ASCII characters. In this case, you have either to use spellcheck.q, or to implement your own QueryConverter.

spellcheck.build

Create the dictionary for use by the SolrSpellChecker. In typical applications, one needs to build the dictionary before using it. However, it may not always be necessary as it is possible to setup the spellchecker with a dictionary that already exists.

spellcheck.reload

Reload the spell checker. Depends on the implementation of SolrSpellChecker.reload() but usually means reloading the dictionary

spellcheck.dictionary

The name of the spellchecker to use. This defaults to "default". Can be used to invoke a specific spellchecker on a per request basis.

spellcheck.count

The maximum number of suggestions to return. Note that this value also limits the number of candidates considered as suggestions. You might need to increase this value to make sure you always get the best suggestion, even if you plan to only use the first item in the list.

spellcheck.alternativeTermCount

The maximum number of suggestions to return for terms that exist in the index (Document Frequency > 0). Specifying this instructs the spellchecker to try and make suggestions for every term in the query. This differs from the "spellcheck.onlyMorePopular" option in that suggested terms need not be "more popular". Also, if used with "spellcheck.collate" collations may be built using the user's original query terms (whereas "spellcheck.onlyMorePopular" will try to correct every term when building collations). ⚠️ Solr4.0 See https://issues.apache.org/jira/browse/SOLR-2585

spellcheck.onlyMorePopular

Only return suggestions that result in more hits for the query than the existing query. Note that even if the given query term is correct (i.e. present in the index), a more popular suggestion will be returned (if one exists).

spellcheck.maxResultsForSuggest

The maximum number of results the query can return while still triggering spelling suggestions (and collations, if using "spellcheck.collate"). Suggestions will not be generated if the query returns more results than this value. When using "spellcheck.extendedResults", this value is also the threshold for determining if the "correctlySpelled" flag is false. (If "spellcheck.maxResultsForSuggest" is not specified, the default behavior is to generate suggestions and to report "correctlySpelled" as "false" if at least 1 term is not in the index (Document Frequency == 0) regardless of the number of results returned.) This parameter is especially useful in conjunction with "spellcheck.alternativeTermCount" to generate "Did You mean?"-style suggestions for low hit-count queries. ⚠️ Solr4.0 See https://issues.apache.org/jira/browse/SOLR-2585

spellcheck.extendedResults

Provide additional information about the suggestion, such as the frequency in the index.

spellcheck.collate

A collation is the original query string with the best suggestions for each term replaced in it. If spellcheck.collate is true, Solr will take the best suggestion for each token (if it exists) and construct a new query from the suggestions. For example, if the input query was "jawa class lording" and the best suggestion for "jawa" was "java" and "lording" was "loading", then the resulting collation would be "java class loading". The top suggestions are used, but no attempt is made to ensure the collation, if re-run by the client, will return any results.

⚠️ Solr4.0 Solr3.1 See https://issues.apache.org/jira/browse/SOLR-2010

spellcheck.collate can guarantee that collations will return results if re-run by the client (applying original fq params also). This is especially helpful when there is more than one correction per query. There is also an option to get multiple collation suggestions and an expanded response format. The following three parameters enable this functionality:

spellcheck.maxCollations

The maximum number of collations to return. Default=1. Ignored if "spellcheck.collate" is false. ⚠️ Solr4.0 Solr3.1

spellcheck.maxCollationTries

The maximum # of collation possibilities to try before giving up. Lower values ensure better performance. Higher values may be necessary to find a collation that can return results. Default is 0 (do not check collations). Ignored if "spellcheck.collate" is false. ⚠️ Solr4.0 Solr3.1

spellcheck.maxCollationEvaluations

The maximum number of word correction combinations to rank and evaluate prior to deciding which collation candidates to test against the index. This is a performance safety-net in cases a user enters a query with many misspelled words. The default is 10,000 combinations which should work well in most situations. Ignored if "spellcheck.collate" is false. ⚠️ Solr4.0 Solr3.3

spellcheck.collateParam.XX

For use with "spellcheck.maxCollationTries". Override the named parameter (substitute XX). For instance, if user's query uses dismax/edismax and a low "mm" value was specified (such as 1), it might be desired to requre 100% of the query terms to match when testing collations. In this case, specify "spellcheck.collateParam.mm=100%". ⚠️ Solr4.0 See https://issues.apache.org/jira/browse/SOLR-3211

spellcheck.collateExtendedResults

If true, returns an expanded response format detailing collations found. default is false. Ignored if "spellcheck.collate" is false. Following is an example of the extended output for the misspelled query **Title:(hopq AND faill)** ⚠️ Solr4.0 Solr3.1

```
<lst name="collation">
    <str name="collationQuery">Title:(hope AND faith)</str>
    <int name="hits">2</int>
    <lst name="misspellingsAndCorrections">
        <str name="hopq">hope</str>
        <str name="faill">faith</str>
    </lst>
</lst>
<lst name="collation">
    <str name="collationQuery">Title:(chops AND all)</str>
    <int name="hits">1</int>
    <lst name="misspellingsAndCorrections">
        <str name="hopq">chops</str>
        <str name="faill">all</str>
    </lst>
</lst>
```

spellcheck.collateMaxCollectDocs

Specify the maximum number of documents SpellCheckComponent should collect when testing potential Collations against the index. The default (0) indicates that all documents should be collected, resulting in exact hit-counts. Otherwise an estimation is provided as a performance optimization in cases where exact hit-counts are unnecessary. Also, when "spellcheck.collateExtendedResults" is false, this optimization is always made (as if 1 had been specified here). ⚠️ Solr4.4 Solr5.0

spellcheck.accuracy

⚠️ Solr4.0 Solr3.1 See https://issues.apache.org/jira/browse/LUCENE-2608

Pass in an accuracy value to be used by the spell checking implementation to decide whether a result is worthwhile or not. Defaults to Float.MIN_VALUE.

spellcheck.<DICT_NAME>.key

⚠️ Solr4.0 Solr3.1 See https://issues.apache.org/jira/browse/LUCENE-2608

Pass in a key/value pair to the implementation. This key/value pair is passed through to the implementation in a SolrParams class. The value that is passed through is just key=value (in other words, spellcheck.<DICT_NAME>. is stripped off)

Example: Given a dictionary called foo, spellcheck.foo.myKey=myValue would result in myKey=myValue being passed through to the implementation handling the dictionary foo.

Use in the Solr Example

The Solr example (in solr/example) comes with a preconfigured SearchComponent and an associated RequestHandler for demonstration purposes. See the example solrconfig.xml (solr/example/solr/conf/solrconfig.xml) for setup parameters.

Example Requests

A simple result using the spellcheck.q parameter. Note the spellcheck.build=true which is needed only once to build the index. It should not be specified with for each request.

```
http://localhost:8983/solr/spell?q=*:*&spellcheck.build=true&spellcheck.q=delll%20ultrashar&spellcheck=true
```

```
<lst name="spellcheck">
        <lst name="suggestions">
                <lst name="delll">
                        <int name="numFound">1</int>
                        <int name="startOffset">0</int>
                        <int name="endOffset">5</int>
                        <arr name="suggestion">
                                <str>dell</str>
                        </arr>
                </lst>
                <lst name="ultrashar">
                        <int name="numFound">1</int>
                        <int name="startOffset">6</int>
                        <int name="endOffset">15</int>
                        <arr name="suggestion">
                                <str>ultrasharp</str>
                        </arr>
                </lst>
        </lst>
</lst>
```

Extended Results

The spellcheck.extendedResults=true parameter provides frequency of each original term in the index (origFreq) as well as the frequency of each suggestion in the index (frequency).

***NOTE**: This result format differs from the non-extended one as the returned suggestion for a word is actually an array of lists, where each list holds the suggested term and its frequency.* ⚠️ Solr1.4

```
http://localhost:8983/solr/spell?q=*:*&spellcheck.q=delll+ultrashar&spellcheck=true&spellcheck.
extendedResults=true
```

```
<lst name="spellcheck">
  <lst name="suggestions">
    <lst name="delll">
      <int name="numFound">1</int>
      <int name="startOffset">0</int>
      <int name="endOffset">5</int>
      <int name="origFreq">0</int>
      <arr name="suggestion">
        <lst>

          <str name="word">dell</str>
          <int name="freq">2</int>
        </lst>
      </arr>
    </lst>
    <lst name="ultrashar">
      <int name="numFound">1</int>

      <int name="startOffset">6</int>
      <int name="endOffset">15</int>
      <int name="origFreq">0</int>
      <arr name="suggestion">
        <lst>
          <str name="word">ultrasharp</str>
          <int name="freq">2</int>

        </lst>
      </arr>
    </lst>
    <bool name="correctlySpelled">false</bool>
  </lst>
</lst>
```

Collate Results

Adding the spellcheck.collate=true parameter returns a query with the misspelled terms replaced by the top suggestions. Note that the non-spellcheckable terms such as those for range queries, prefix queries etc. are detected and excluded for spellchecking. Such non-spellcheckable terms are preserved in the collated output so that the original query can be run again, as is.

```
http://localhost:8983/solr/spell?q=price:[80 TO 100] delll ultrashar&spellcheck=true&spellcheck.
extendedResults=true&spellcheck.collate=true
```

```
<lst name="spellcheck">
  <lst name="suggestions">
    <lst name="delll">
      <int name="numFound">1</int>
      <int name="startOffset">18</int>
      <int name="endOffset">23</int>
      <int name="origFreq">0</int>
      <arr name="suggestion">
        <lst>
          <str name="word">dell</str>
          <int name="freq">2</int>
        </lst>
      </arr>
    </lst>
    <lst name="ultrashar">
      <int name="numFound">1</int>
      <int name="startOffset">24</int>
      <int name="endOffset">33</int>
      <int name="origFreq">0</int>
      <arr name="suggestion">
        <lst>
          <str name="word">ultrasharp</str>
          <int name="freq">2</int>
        </lst>
      </arr>
    </lst>
    <bool name="correctlySpelled">false</bool>
    <str name="collation">price:[80 TO 100] dell ultrasharp</str>
  </lst>
</lst>
```

Implementing a new java SolrSpellChecker

⚠️ :TODO: ⚠️ HOOK in links to Javadocs.

The SolrSpellChecker class provides an abstract base class for defining common spelling constructs for use in the SpellCheckComponent. Implementing classes need to define the following methods:

1. reload - How to reload the dictionary/spell checker. This method is called when the application knows there are changes to the dictionary and that they should be loaded.
2. build - Create the appropriate spelling resources. Also called when the resources needs to be rebuilt. Not all implementations may need to implement this. For instance, an implementation may always use the same underlying resources and they are immutable. The Lucene IndexBased SpellChecker, on the other hand, actually creates the appropriate underlying dictionary from the specified index.
3. getSuggestions(Collection<Token> tokens, IndexReader reader, int count, boolean onlyMorePopular, boolean extendedResults) - The main method called for returning suggestions. See the javadocs for more explanation.

Implementing a QueryConverter

The QueryConverter is an abstract base class defining a method for converting input "raw" queries into a set of tokens for spell checking. It is used to "parse" the CommonParams.Q (the input query) and convert it to tokens. It is only invoked for the CommonParams.Q parameter, and not the "spellcheck.q" parameter. Systems that use their own query parser or those that find issues with the basic implementation will want to implement their own QueryConverter. Instead of using the provided implementation (SpellingQueryConverter), they should override the appropriate methods on the SpellingQueryConverter in their custom QueryConverter and register it in the solrconfig.xml via:

```
<queryConverter name="queryConverter" class="org.apache.solr.spelling.SpellingQueryConverter"/>
```

The existing converter uses a relatively simple Regex to extract out the basic query terms from a query and create tokens from them.

Distributed Search Support

⚠️ Solr3.1

SpellCheckComponent now supports distributed setups. If you are using SpellCheckComponent on a request handler other than "/select", then you need to provide the following two parameters:

- "shards" - See DistributedSearch
- "shards.qt" - Signals Solr that requests to shards should be sent to a request handler given by this parameter. Use shards.qt=/spell when making the request if your request handler is "/spell".

For example:

```
http://solr:8983/solr/select?q=*:*&spellcheck=true&spellcheck.build=true&spellcheck.q=toyata&qt=spell&shards.qt=
/spell&shards=solr-shard1:8983/solr,solr-shard2:8983/solr
```

If SpellCheckComponent is added to the /select request handler, then the "shards.qt" parameter is not required.

In case of a distributed request to SpellCheckComponent, the shards are requested for at least five suggestions even if "spellcheck.count" is less than five. Once the suggestions are collected, they are ranked by the configured distance measure (default is Levenstein Distance) and then by aggregate frequency.

History

For discussion of the development of this feature, see SOLR-572.