



Apache
Solr  Query Parsing

Presented by Erik Hatcher
27 February 2013

Description

Interpreting what the user meant and what they ideally would like to find is tricky business. This talk will cover useful tips and tricks to better leverage and extend **Solr's** analysis and query parsing capabilities to more richly parse and interpret user queries.

Abstract

In this talk, **Solr's built-in query parsers** will be detailed including when and how to use them. Solr has nested query parsing capability, allowing for multiple query parsers to be used to generate a single query. The nested query parsing feature will be described and demonstrated. In many domains, e-commerce in particular, parsing queries often means interpreting which entities (e.g. products, categories, vehicles) the user likely means; this talk will conclude with techniques to achieve richer query interpretation.

Query parsers in Solr

```
public abstract class QParserPlugin implements NamedListInitializedPlugin {
    /** internal use - name of the default parser */
    public static String DEFAULT_QTYPE = LuceneQParserPlugin.NAME;

    /** internal use - name to class mappings of builtin parsers */
    public static final Object[] standardPlugins = {
        LuceneQParserPlugin.NAME, LuceneQParserPlugin.class,
        OldLuceneQParserPlugin.NAME, OldLuceneQParserPlugin.class,
        FunctionQParserPlugin.NAME, FunctionQParserPlugin.class,
        PrefixQParserPlugin.NAME, PrefixQParserPlugin.class,
        BoostQParserPlugin.NAME, BoostQParserPlugin.class,
        DisMaxQParserPlugin.NAME, DisMaxQParserPlugin.class,
        ExtendedDismaxQParserPlugin.NAME, ExtendedDismaxQParserPlugin.class,
        FieldQParserPlugin.NAME, FieldQParserPlugin.class,
        RawQParserPlugin.NAME, RawQParserPlugin.class,
        TermQParserPlugin.NAME, TermQParserPlugin.class,
        NestedQParserPlugin.NAME, NestedQParserPlugin.class,
        FunctionRangeQParserPlugin.NAME, FunctionRangeQParserPlugin.class,
        SpatialFilterQParserPlugin.NAME, SpatialFilterQParserPlugin.class,
        SpatialBoxQParserPlugin.NAME, SpatialBoxQParserPlugin.class,
        JoinQParserPlugin.NAME, JoinQParserPlugin.class,
        SurroundQParserPlugin.NAME, SurroundQParserPlugin.class,
    };
};
```

Query Parsers in Solr

```
public abstract class QParserPlugin implements NamedListInitializedPlugin {
    /** internal use - name of the default parser */
    public static String DEFAULT_QTYPE = LuceneQParserPlugin.NAME;

    /** internal use - name to class mappings of builtin parsers */
    public static final Object[] standardPlugins = {
        "lucene", LuceneQParserPlugin.class,
        "lucenePlusSort", OldLuceneQParserPlugin.class,
        "func", FunctionQParserPlugin.class,
        "prefix", PrefixQParserPlugin.class,
        "boost", BoostQParserPlugin.class,
        "dismax", DisMaxQParserPlugin.class,
        "edismax", ExtendedDisMaxQParserPlugin.class,
        "field", FieldQParserPlugin.class,
        "raw", RawQParserPlugin.class,
        "term", TermQParserPlugin.class,
        "query", NestedQParserPlugin.class,
        "frange", FunctionRangeQParserPlugin.class,
        "geofilt", SpatialFilterQParserPlugin.class,
        "bbox", SpatialBoxQParserPlugin.class,
        "join", JoinQParserPlugin.class,
        "surround", SurroundQParserPlugin.class,
    };
};
```

What's new in 4.x?

- Surround query parser
- `_query_:"{!....}"` ugliness now unnecessary
 - just use nested `{!....}` expressions
- Coming to 4.2: "switch" query parser

"lucene"-syntax query parser

- FieldType awareness
 - range queries, numerics
 - allows date math
 - reverses wildcard terms, if indexing used ReverseWildcardFilter
- Magic fields
 - `_val_`: function query injection
 - `_query_`: nested query, to use a different query parser
- Multi-term analysis (`type="multiterm"`)
 - Analyzes prefix, wildcard, regex expressions to normalize diacritics, lowercase, etc
 - If not explicitly defined, all MultiTermAwareComponent's from query analyzer are used, or KeywordTokenizer for effectively no analysis

dismax

- Simple constrained syntax
 - "supports phrases" +requiredTerms -prohibitedTerms loose terms
- Spreads terms across specified query fields (qf) and entire query string across phrase fields (pf)
 - with field-specific boosting
 - and explicit and implicit phrase slop
 - scores each document with the maximum score for that document as produced by any subquery; primary score associated with the highest boost, not the sum of the field scores (as BooleanQuery would give)
- Minimum match (mm) allows query fields gradient between AND and OR
 - some number of terms must match, but not all necessarily, and can vary depending on number of actual query terms
- Additive boost queries (bq) and boost functions (bf)
- Debug output includes parsed boost and function queries

Specifying the query parser

- `defType=parser_name`
 - defines main query parser
- `{!parser_name local=param...}expression`
 - Can specify parser per query expression
- These are equivalent:
 - `q=ApacheCon NA 2013&defType=dismax&mm=2&qf=name`
 - `q={!dismax qf=name mm=2}ApacheCon NA 2013`
 - `q={!dismax qf=name mm=2 v='ApacheCon NA 2013'}`

Local Parameter Substitution

/document?id=13

```
<requestHandler name="/document" class="solr.SearchHandler">
  <lst name="invariants">
    <str name="q">{!term f=id v=$id}</str>
    <str name="rows">1</str>
  </lst>
  <arr name="components">
    <str>query</str>
    <str>highlight</str>
    <str>debug</str>
  </arr>
</requestHandler>
```

Nested Query Parsing

- Leverages the "lucene" query parser's `_query_{!...}` trick
- Example:
 - `q={!dismax qf='title^2 body' v=$user_query} AND
 {!dismax qf='keywords^5 description^2' v=$topic}`
 - `&user_query=ApacheCon NA 2013`
 - `&topic=events`
- Setting the complex nested `q` parameter in a request handler can make the client request lean and clean
 - And even `qf` and other parameters can be substituted:
 - `{!dismax qf=$title_qf pf=$title_pf v=$title_query}`
 - `&title_qf=title^5 subtitle^2...`
- Real world example, Stanford University Libraries:
 - `http://searchworks.stanford.edu/advanced`
 - Insanely complex sets of nested `dismax`'s and `qf/pf` settings

edismax: extended dismax

- "An advanced multi-field query parser based on the dismax parser"
 - Handles "lucene" syntax as well as dismax features
- Fields available to user may be limited (uf)
 - including negations and dynamic fields, e.g. uf=* -cost -timestamp
- Shingles query into 2 and 3 term phrases
 - Improves quality of results when query contains terms across multiple fields
 - pf2/pf3 and ps2/ps3
 - removes stop words from shingled phrase queries
- multiplicative "boost" functions
- Additional features
 - Query comprised entirely of "stopwords" optionally allowed
 - if indexed, but query analyzer is set to remove them
 - Allow "lowercaseOperators" by default; or/OR, and/AND

term query parser

- FieldType aware, no analysis
 - converts to internal representation automatically
- "raw" query parser is similar
 - though raw parser is not field type aware; no internal representation conversion
- Best practice for filtering on single facet value
 - `fq={!term f=facet_field}crazy:value :)`
 - no query string escaping needed; but of course still need URL encoding when appropriate

prefix query parser

- No field type awareness
- `{!prefix f=field_name}prefixValue`
 - Similar to Lucene query parser
`field_name:prefixValue*`
 - Solr's "lucene" query parser has multiterm analysis capability, but the prefix query parser does not analyze

boost query parser

- Multiplicative to wrapped query score
 - Internally used by edismax "boost"
- `{!boost b=recip(ms(NOW,mydatefield), 3.16e-11,1,1)}foo`

field query parser

- Same as handling of field:"Some Text" clause by Solr's "lucene" query parser
- FieldType aware
 - TermQuery generated, unless field type has special handling
- TextField
 - PhraseQuery: if multiple tokens in different positions
 - MultiPhraseQuery: if multiple tokens share some positions
 - BooleanQuery: if multiple terms all in same position
 - TermQuery: if only a single token
- Other types that handle field queries specially:
 - currency, spatial types (point, latlon, etc)
 - `{!field f=location}49.25,8.883333`

surround query parser

- Creates Lucene SpanQuery's for fine-grained proximity matching, including use of wildcards
- Uses infix and prefix notation
 - infix: AND/OR/NOT/nW/nN/()
 - prefix: AND/OR/nW/nN
 - Supports Lucene query parser basics
 - field:value, boost^5, wild?c*rd, prefix*
 - Proximity operators:
 - N: ordered
 - W: unordered
- No analysis of clauses
 - requires user or search client to lowercase, normalize, etc
- Example:
 - q={!surround}Apache* 4w Portland

join query parser

- Pseudo-join
 - Field values from inner result set used to map to another field to select final result set
 - No information from inner result set carries to final result set, such as scores or field values (it's not SQL!)
- Can join from another local Solr core
 - Allows for different types of entities to be indexed in separate indexes altogether, modeled into clean schemas
 - Separate cores can scale independently, especially with commit and warming issues
- Syntax:
 - `{!join from=... to=... [fromIndex=core_name]}query`
- For more information:
 - Yonik's Lucene Revolution 2011 presentation: <http://vimeo.com/25015101>
 - <http://wiki.apache.org/solr/Join>

spatial query parsers

- Operates on geohash, latlon, and point types
- geofilt
 - Exact distance filtering
 - `fq={!geofilt sfield=location pt=10.312,-20.556 d=3.5}`
- bbox
 - Alternatively use a range query:
 - `fq=location:[45,-94 TO 46,-93]`
- Can use in conjunction with `geodist()` function
 - Sorting:
 - `sort=geodist() asc`
 - Returning distance:
 - `fl=_dist_:geodist()`

frange: function range

- Match a field term range, textual or numeric
- Example:
 - fq={!frange l=0 u=2.2}
sum(user_ranking,editor_ranking)

switch query parser

- acts like a "switch/case" statement

- Example:

- fq={!switch
 case.all='*:*'
 case.yes='inStock:true'
 case.no='inStock:false'
 v=\$in_stock}

- &in_stock=yes

- Solr 4.2+

PostFilter

- Query's implementing PostFilter interface consulted after query and all other filters have narrowed documents for consideration
- Queries supporting PostFilter
 - frange, geofilt, bbox
- Enabled by setting `cache=false` and `cost >= 100`
 - Example:
 - `fq={!frange l=5 cache=false cost=200}div(log(popularity),sqrt(geodist()))`
- More info:
 - Advanced filter caching
 - <http://searchhub.org/2012/02/10/advanced-filter-caching-in-solr/>
 - Custom security filtering
 - <http://searchhub.org/2012/02/22/custom-security-filtering-in-solr/>

Phonetic, Stem, Synonym

- Users tend to expect loose matching
 - but with "more exact" matches ranked higher
- Various mechanisms for loosening matching:
 - Phonetic sounds-like: cat/kat, similar/similer
 - Stemming: search/searches/searched/searching
 - Synonyms: cat/feline, dog/canine
- Distinguish ranking between exact and looser matching:
 - copyField original to a new (unstored, yet indexed) field with desired looser matching analysis
 - query across original field and looser field, with higher boosting for original field
 - `/select?q=ApatchyCon&defType=dismax&qf=name^5 name_phonetic`

Suggest things, not strings

- Model It As You Need It
 - Leverage Lucene's Document/Field/Query/score & sort & highlight
- Example 1: Selling automobile parts
 - Exact year/make/model is needed to pick the right parts
 - Suggest a vehicle as user types
 - from the main parts index: tricky, requires lots of special fields and analysis tricks and even then you're suggesting fields from "parts"
 - Another (better?) approach: model vehicles as a separate core, "search" when suggesting, return documents, not field terms
- Example 2: Technical Conferences
 - `/select?q=Con&wt=csv&fl=name`
 - Lucene EuroCon
 - ApacheCon

Query parsing and relevancy

- The query is the formula that determines each document's score
- Tuning is about what your application needs
 - Build tests using your corpus and real-world queries and ranking expectations
 - Re-run tests frequently/continuously as query parameters are tweaked
- Tooling, currently, is mostly in-house custom
 - but that's changing, stay tuned!

Development/troubleshooting

- Analysis

- /analysis/field

- ?analysis.fieldname=name
- &analysis.fieldvalue=NA ApacheCon 2013
- &q=apachecon
- &analysis.showmatch=true

- Also /analysis/document

- admin UI analysis tool

- Query Parsing

- &debug=query

- Relevancy

- &debug=results
 - shows scoring explanations

```
- <lst>
  <str name="text">apachecon</str>
  <str name="raw_bytes">[61 70 61 63 68 65 63 6f 6e]</str>
  <bool name="match">true</bool>
  <int name="position">2</int>
- <arr name="positionHistory">
  <int>2</int>
  <int>2</int>
  <int>2</int>
</arr>
  <int name="start">3</int>
  <int name="end">12</int>
  <str name="type"><ALPHANUM></str>
</lst>
```

Future of Solr query parsing

- JSON query parser
- XML query parser
- PayloadTermQuery parser

JSON query parser

- <https://issues.apache.org/jira/browse/SOLR-4351>
- Current patch enables these:
 - {'term':{'id':'13'}}
 - {'field':{'text':'ApacheCon'}}
 - {'frange':{'v':'mul(rating,2)', 'l':20,'u':24}}
 - {'join':{'from':'book_id', 'to':'id', 'v':{'term':{'text':'search'}}}}

XML query parser

- Will allow a rich query "tree"
- Parameters will fill in variables in a server-side XSLT query tree definition, or can provide full query tree
- Useful for "advanced" query, multi-valued, input
- <https://issues.apache.org/jira/browse/SOLR-839>

Payload term query parser

- Solr supports indexing payload data on terms using `DelimitedPayloadTokenFilter`, but currently no support for querying with payloads
- Requires custom Similarity implementation to provide score factor for payload data
- Allows index-time weighting of terms
 - e.g. `bold words` weighted higher
- <https://issues.apache.org/jira/browse/SOLR-1485>

BlockJoinQuery

- <https://issues.apache.org/jira/browse/SOLR-3076>
- Lucene provides a way to index a hierarchical "block" of documents and query it using `ToParentBlockJoinQuery` and `ToChildBlockJoinQuery`
 - Indexing a block is not yet supported by Solr
- Example use case: What books greater than 100 pages have paragraphs containing "information retrieval"?

