Package ‘solr’

August 29, 2016

Title General Purpose R Interface to Solr

Description Provides a set of functions for querying and parsing
data from Solr endpoints (local and remote), including search, faceting,
highlighting, stats, and ‘more like this’.

Version 0.1.6
Date 2015-09-17
License MIT + file LICENSE

URL https://github.com/ropensci/solr

BugReports http://www.github.com/ropensci/solr/issues

LazyLoad true

LazyData true

VignetteBuilder knitr

Imports methods, utils, plyr, httr, XML, assertthat, rjson

Suggests testthat, roxygen2, knitr

NeedsCompilation no

Author Scott Chamberlain [aut, cre]

Maintainer Scott Chamberlain <myrmecocystus@gmail.com>

Repository CRAN

Date/Publication 2015-09-17 22:50:16

R topics documented:

  solr-package .......................................................... 2
  collectargs ......................................................... 2
  is.sr_facet .......................................................... 3
  makemultiargs ..................................................... 3
  solr_all ............................................................. 4
  solr_facet .......................................................... 6
  solr_group .......................................................... 12
  solr_highlight ..................................................... 14
Description

The solr package is an R interface to Solr. Currently this package only does the getting data part, not writing data, but if you want data writing capability do speak up and/or send a pull request.

Details

There are currently three main functions:

solr_search General search
solr_facet Faceting only (w/o general search)
solr_highlight Highlighting only (w/o general search)

See the vignettes for help:

- solr_localsetup
- solr_vignette

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>

collectargs

Function to make a list of args passing arg names through multiargs function.

Usage

collectargs(x)

Arguments

x Value
is.sr_facet

Test for sr_facet class

Test for sr_high class
Test for sr_search class

Usage

is.sr_facet(x)

is.sr_high(x)

is.sr_search(x)

Arguments

x Input

makemultiargs

Function to make make multiple args of the same name from a single input with length > 1

Description

Function to make make multiple args of the same name from a single input with length > 1

Usage

makemultiargs(x)

Arguments

x Value
Solr search.

Description

Solr search.

Usage

solr_all(q = "*:*", sort = NULL, start = 0, rows = NULL, 
pageDoc = NULL, pageScore = NULL, fq = NULL, fl = NULL, 
defType = NULL, timeAllowed = NULL, qt = NULL, wt = "json", 
NOW = NULL, TZ = NULL, echoHandler = NULL, echoParams = NULL, 
key = NULL, base = NULL, callopts = list(), raw = FALSE, 
parsetype = "df", concat = "","", ... , verbose = TRUE)

Arguments

q Query terms, defaults to '*:*', or everything.
sort Field to sort on. You can specify ascending (e.g., score desc) or descending (e.g., 
score asc), sort by two fields (e.g., score desc, price asc), or sort by a function 
(e.g., sum(x_f, y_f) desc, which sorts by the sum of x_f and y_f in a descending order).
start Record to start at, default to beginning.
rows Number of records to return. Defaults to 10.
pageDoc If you expect to be paging deeply into the results (say beyond page 10, assum-
ing rows=10) and you are sorting by score, you may wish to add the pageDoc 
and pageScore parameters to your request. These two parameters tell Solr (and 
Lucene) what the last result (Lucene internal docid and score) of the previous 
page was, so that when scoring the query for the next set of pages, it can ignore 
any results that occur higher than that item. To get the Lucene internal doc id, 
you will need to add [docid] to the &fl list. e.g., q=*:*&start=10&pageDoc=5&pageScore=1.345&fl=[docid]
pageScore See pageDoc notes.
fq Filter query, this does not affect the search, only what gets returned
fl Fields to return
defType Specify the query parser to use with this request.
timeAllowed The time allowed for a search to finish. This value only applies to the search and 
not to requests in general. Time is in milliseconds. Values <= 0 mean no time 
restriction. Partial results may be returned (if there are any).
qt Which query handler used.
wt Data type returned, defaults to 'json'
NOW Set a fixed time for evaluating Date based expressions
TZ Time zone, you can override the default.
solr_all

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>echoHandler</td>
<td>If the echoHandler parameter is true, Solr places the name of the handle used in the response to the client for debugging purposes.</td>
</tr>
<tr>
<td>echoParams</td>
<td>The echoParams parameter tells Solr what kinds of Request parameters should be included in the response for debugging purposes, legal values include:</td>
</tr>
<tr>
<td></td>
<td>• none - don’t include any request parameters for debugging</td>
</tr>
<tr>
<td></td>
<td>• explicit - include the parameters explicitly specified by the client in the request</td>
</tr>
<tr>
<td></td>
<td>• all - include all parameters involved in this request, either specified explicitly by the client, or implicit because of the request handler configuration.</td>
</tr>
<tr>
<td>key</td>
<td>API key, if needed.</td>
</tr>
<tr>
<td>base</td>
<td>URL endpoint.</td>
</tr>
<tr>
<td>callopts</td>
<td>Call options passed on to httr::GET</td>
</tr>
<tr>
<td>raw</td>
<td>(logical) If TRUE, returns raw data in format specified by wt param</td>
</tr>
<tr>
<td>parseType</td>
<td>(character) One of 'list' or 'df'</td>
</tr>
<tr>
<td>concat</td>
<td>(character) Character to concatenate elements of longer than length 1. Note that this only works reliably when data format is json (wt='json'). The parsing is more complicated in XML format, but you can do that on your own.</td>
</tr>
<tr>
<td>...</td>
<td>Further args.</td>
</tr>
<tr>
<td>verbose</td>
<td>If TRUE (default) the url call used printed to console.</td>
</tr>
</tbody>
</table>

**Value**

XML, JSON, a list, or data.frame

**References**


**See Also**

solr_highlight, solr_facet

**Examples**

```r
## Not run:
url <- 'http://api.plos.org/search'
solr_all(q='*:*', rows=2, fl='id', base=url)
## End(Not run)
```
solr_facet

Do faceted searches, outputing facets only.

Description

Do faceted searches, outputing facets only.

Usage

solr_facet(q = "*:*", facet.query = NA, facet.field = NA,
    facet.prefix = NA, facet.sort = NA, facet.limit = NA,
    facet.offset = NA, facet.mincount = NA, facet.missing = NA,
    facet.method = NA, facet.enum.cache.minDf = NA, facet.threads = NA,
    facet.date = NA, facet.date.start = NA, facet.date.end = NA,
    facet.date.gap = NA, facet.date.hardend = NA, facet.date.other = NA,
    facet.date.include = NA, facet.range = NA, facet.range.start = NA,
    facet.range.end = NA, facet.range.gap = NA, facet.range.hardend = NA,
    facet.range.other = NA, facet.range.include = NA, start = NA,
    rows = NA, key = NA, base = NA, wt = "json", raw = FALSE,
    callopts = list(), verbose = TRUE, ...)

Arguments

q
    Query terms. See examples.

facet.query
    This param allows you to specify an arbitrary query in the Lucene default syntax
to generate a facet count. By default, faceting returns a count of the unique
terms for a "field", while facet.query allows you to determine counts for arbitrary
terms or expressions. This parameter can be specified multiple times to indicate
that multiple queries should be used as separate facet constraints. It can be
particularly useful for numeric range based facets, or prefix based facets – see
example below (i.e. price:[* TO 500] and price:[501 TO *]).

facet.field
    This param allows you to specify a field which should be treated as a facet. It
will iterate over each Term in the field and generate a facet count using that Term
as the constraint. This parameter can be specified multiple times to indicate
multiple facet fields. None of the other params in this section will have any
effect without specifying at least one field name using this param.

facet.prefix
    Limits the terms on which to facet to those starting with the given string prefix.
    Note that unlike fq, this does not change the search results – it merely reduces the
    facet values returned to those beginning with the specified prefix. This parameter
can be specified on a per field basis.

facet.sort
    See Details.

facet.limit
    This param indicates the maximum number of constraint counts that should be
    returned for the facet fields. A negative value means unlimited. Default: 100.
    Can be specified on a per field basis.
facet.offset  This param indicates an offset into the list of constraints to allow paging. Default: 0. This parameter can be specified on a per field basis.

facet.mincount  This param indicates the minimum counts for facet fields should be included in the response. Default: 0. This parameter can be specified on a per field basis.

facet.missing  Set to "true" this param indicates that in addition to the Term based constraints of a facet field, a count of all matching results which have no value for the field should be computed. Default: FALSE. This parameter can be specified on a per field basis.

facet.method  See Details.

facet.enum.cache.minDF  This param indicates the minimum document frequency (number of documents matching a term) for which the filterCache should be used when determining the constraint count for that term. This is only used when facet.method=enum method of faceting. A value greater than zero will decrease memory usage of the filterCache, but increase the query time. When faceting on a field with a very large number of terms, and you wish to decrease memory usage, try a low value of 25 to 50 first. Default: 0, causing the filterCache to be used for all terms in the field. This parameter can be specified on a per field basis.

facet.threads  This param will cause loading the underlying fields used in faceting to be executed in parallel with the number of threads specified. Specify as facet.threads=\# where \# is the maximum number of threads used. Omitting this parameter or specifying the thread count as 0 will not spawn any threads just as before. Specifying a negative number of threads will spin up to Integer.MAX_VALUE threads. Currently this is limited to the fields, range and query facets are not yet supported. In at least one case this has reduced warmup times from 20 seconds to under 5 seconds.

facet.date  Specify names of fields (of type DateField) which should be treated as date facets. Can be specified multiple times to indicate multiple date facet fields.

facet.date.start  The lower bound for the first date range for all Date Faceting on this field. This should be a single date expression which may use the DateMathParser syntax. Can be specified on a per field basis.

facet.date.end  The minimum upper bound for the last date range for all Date Faceting on this field (see facet.date.hardend for an explanation of what the actual end value may be greater). This should be a single date expression which may use the DateMathParser syntax. Can be specified on a per field basis.

facet.date.gap  The size of each date range expressed as an interval to be added to the lower bound using the DateMathParser syntax. Eg: facet.date.gap=+1DAY (+1DAY). Can be specified on a per field basis.

facet.date.hardend  A Boolean parameter instructing Solr what to do in the event that facet.date.gap does not divide evenly between facet.date.start and facet.date.end. If this is true, the last date range constraint will have an upper bound of facet.date.end; if false, the last date range will have the smallest possible upper bound greater then facet.date.end such that the range is exactly facet.date.gap wide. Default: FALSE. This parameter can be specified on a per field basis.
facet.date.other
    See Details.
facet.date.include
    See Details.
facet.range
    Indicates what field to create range facets for. Example: facet.range=price&facet.range=age

facet.range.start
    The lower bound of the ranges. Can be specified on a per field basis. Example:
    f.pricefacet.range.start=0.0&f.agefacet.range.start=10

facet.range.end
    The upper bound of the ranges. Can be specified on a per field basis. Example:
    f.pricefacet.range.end=1000.0&f.agefacet.range.start=99

facet.range.gap
    The size of each range expressed as a value to be added to the lower bound. For date fields, this should be expressed using the DateMathParser syntax. Can be specified on a per field basis. Example: f.pricefacet.range.gap=100&f.agefacet.range.gap=10

facet.range.hardend
    A Boolean parameter instructing Solr what to do in the event that facet.range.gap does not divide evenly between facet.range.start and facet.range.end. If this is true, the last range constraint will have an upper bound of facet.range.end; if false, the last range will have the smallest possible upper bound greater then facet.range.end such that the range is exactly facet.range.gap wide. Default: FALSE. This parameter can be specified on a per field basis.

facet.range.other
    See Details.
facet.range.include
    See Details.

start
    Record to start at, default to beginning.

rows
    Number of records to return.

key
    API key, if needed.

base
    URL endpoint

wt
    (character) Data format to return. One of xml or json (default).

raw
    (logical) If TRUE (default) raw json or xml returned. If FALSE, parsed data returned.

callopts
    Call options passed on to http::GET

verbose
    If TRUE (default) the url call used printed to console.

... Further args, usually per field arguments for faceting.

Details

A number of fields can be specified multiple times, in which case you can separate them by commas, like facet.field='journal,subject'. Those fields are:

- facet.field
- facet.query
- facet.date
- facet.date.other
- facet.date.include
- facet.range
- facet.range.other
- facet.range.include

**Options for some parameters:**

**facet.sort:** This param determines the ordering of the facet field constraints.

- count sort the constraints by count (highest count first)
- index to return the constraints sorted in their index order (lexicographic by indexed term). For terms in the ascii range, this will be alphabetically sorted.

The default is count if facet.limit is greater than 0, index otherwise. This parameter can be specified on a per field basis.

**facet.method:** This parameter indicates what type of algorithm/method to use when faceting a field.

- enum Enumerates all terms in a field, calculating the set intersection of documents that match the term with documents that match the query. This was the default (and only) method for faceting multi-valued fields prior to Solr 1.4.
- fc (Field Cache) The facet counts are calculated by iterating over documents that match the query and summing the terms that appear in each document. This was the default method for single valued fields prior to Solr 1.4.
- fcs (Field Cache per Segment) works the same as fc except the underlying cache data structure is built for each segment of the index individually

The default value is fc (except for BoolField which uses enum) since it tends to use less memory and is faster than the enumeration method when a field has many unique terms in the index. For indexes that are changing rapidly in NRT situations, fcs may be a better choice because it reduces the overhead of building the cache structures on the first request and/or warming queries when opening a new searcher – but tends to be somewhat slower than fc for subsequent requests against the same searcher. This parameter can be specified on a per field basis.

**facet.date.other:** This param indicates that in addition to the counts for each date range constraint between facet.date.start and facet.date.end, counts should also be computed for...

- before All records with field values lower than lower bound of the first range
- after All records with field values greater than the upper bound of the last range
- between All records with field values between the start and end bounds of all ranges
- none Compute none of this information
- all Shortcut for before, between, and after

This parameter can be specified on a per field basis. In addition to the all option, this parameter can be specified multiple times to indicate multiple choices – but none will override all other options.

**facet.date.include:** By default, the ranges used to compute date faceting between facet.date.start and facet.date.end are all inclusive of both endpoints, while the "before" and "after" ranges are not inclusive. This behavior can be modified by the facet.date.include param, which can be any combination of the following options...
• lower All gap based ranges include their lower bound
• upper All gap based ranges include their upper bound
• edge The first and last gap ranges include their edge bounds (ie: lower for the first one, upper for the last one) even if the corresponding upper/lower option is not specified
• outer The "before" and "after" ranges will be inclusive of their bounds, even if the first or last ranges already include those boundaries.
• all Shorthand for lower, upper, edge, outer

This parameter can be specified on a per field basis. This parameter can be specified multiple times to indicate multiple choices.

facet.date.include: This param indicates that in addition to the counts for each range constraint between facet.range.start and facet.range.end, counts should also be computed for...

• before All records with field values lower then lower bound of the first range
• after All records with field values greater then the upper bound of the last range
• between All records with field values between the start and end bounds of all ranges
• none Compute none of this information
• all Shortcut for before, between, and after

This parameter can be specified on a per field basis. In addition to the all option, this parameter can be specified multiple times to indicate multiple choices – but none will override all other options.

facet.range.include: By default, the ranges used to compute range faceting between facet.range.start and facet.range.end are inclusive of their lower bounds and exclusive of the upper bounds. The "before" range is exclusive and the "after" range is inclusive. This default, equivalent to lower below, will not result in double counting at the boundaries. This behavior can be modified by the facet.range.include param, which can be any combination of the following options...

• lower All gap based ranges include their lower bound
• upper All gap based ranges include their upper bound
• edge The first and last gap ranges include their edge bounds (ie: lower for the first one, upper for the last one) even if the corresponding upper/lower option is not specified
• outer The "before" and "after" ranges will be inclusive of their bounds, even if the first or last ranges already include those boundaries.
• all Shorthand for lower, upper, edge, outer

Can be specified on a per field basis. Can be specified multiple times to indicate multiple choices. If you want to ensure you don’t double-count, don’t choose both lower & upper, don’t choose outer, and don’t choose all.

Value

Raw json or xml, or a list of length 4 parsed elements (usually data.frame’s).

References

See http://wiki.apache.org/solr/SimpleFacetParameters for more information on faceting.
See Also

solr_search, solr_highlight, solr_parse

Examples

```r
## Not run:
url <- 'http://api.plos.org/search'

# Facet on a single field
solr_facet(q='*:*', facet.field='journal', base=url)

# Facet on multiple fields
solr_facet(q='alcohol', facet.field=c('journal','subject'), base=url)

# Using mincount
solr_facet(q='alcohol', facet.field='journal', facet.mincount=500, base=url)

# Using facet.query to get counts
solr_facet(q='*:*', facet.field='journal', facet.query=c('cell','bird'), base=url)

# Date faceting
solr_facet(q='*:*', base=url, facet.date=publication_date, facet.date.start='NOW/DAY-5DAYS', facet.date.end='NOW', facet.date.gap='+1DAY')

# Range faceting
solr_facet(q='*:*', base=url, facet.range='counter_total_all', facet.range.start=5, facet.range.end=1000, facet.range.gap=10)

# Range faceting with > 1 field, same settings
solr_facet(q='*:*', base=url, facet.range=c('counter_total_all','alm_twitterCount'), facet.range.start=5, facet.range.end=1000, facet.range.gap=10)

# Range faceting with > 1 field, different settings
solr_facet(q='*:*', base=url, facet.range=c('counter_total_all','alm_twitterCount'), facet.range.start=5, facet.range.end=1000, facet.range.gap=10, f.counter_total_all.facet.range.end=1000, f.counter_total_all.facet.range.start=5, f.alm_twitterCount.facet.range.end=1000, f.alm_twitterCount.facet.range.start=5)

# Get raw json or xml
## json
solr_facet(q='*:*', facet.field='journal', base=url, raw=TRUE)

## xml
solr_facet(q='*:*', facet.field='journal', base=url, raw=TRUE, wt='xml')

# Get raw data back, and parse later, same as what goes on internally if raw=FALSE (Default)
out <- solr_facet(q='*:*', facet.field='journal', base=url, raw=TRUE)
solr_parse(out)
out <- solr_facet(q='*:*', facet.field='journal', base=url, raw=TRUE, wt='xml')
solr_parse(out)
```
Description

Solr grouped search.

Usage

```solr_group
solr_group(q = "*:*", start = 0, rows = NA, sort = NA, fq = NA,
fl = NA, wt = "json", key = NA, group.field = NA, group.limit = NA,
group.offset = NA, group.sort = NA, group.main = NA,
group.ngroups = NA, group.cache.percent = NA, group.query = NA,
group.format = NA, group.func = NA, base = NA, callopts = list(),
raw = FALSE, parsetype = "df", concat = ",", verbose = TRUE, ...)
```

Arguments

- **q**: Query terms, defaults to ":*:", or everything.
- **start**: [number] The offset into the list of groups.
- **rows**: [number] The number of groups to return. Defaults to 10.
- **sort**: How to sort the groups relative to each other. For example, sort=popularity desc will cause the groups to be sorted according to the highest popularity doc in each group. Defaults to "score desc".
- **fq**: Filter query, this does not affect the search, only what gets returned
- **fl**: Fields to return
- **wt**: Data type returned, defaults to 'json'
- **key**: API key, if needed.
- **group.field**: [fieldname] Group based on the unique values of a field. The field must currently be single-valued and must be either indexed, or be another field type that has a value source and works in a function query - such as ExternalFileField. Note: for Solr 3.x versions the field must by a string like field such as StrField or TextField, otherwise a http status 400 is returned.
- **group.limit**: [number] The number of results (documents) to return for each group. Defaults to 1.
- **group.offset**: [number] The offset into the document list of each group.
group.sort  How to sort documents within a single group. Defaults to the same value as the sort parameter.

group.main  (logical) If true, the result of the last field grouping command is used as the main result list in the response, using group.format=simple

group.ngroups  (logical) If true, includes the number of groups that have matched the query. Default is false. <!> Solr4.1 WARNING: If this parameter is set to true on a sharded environment, all the documents that belong to the same group have to be located in the same shard, otherwise the count will be incorrect. If you are using SolrCloud, consider using "custom hashing"

group.cache.percent  [0-100] If > 0 enables grouping cache. Grouping is executed actual two searches. This option caches the second search. A value of 0 disables grouping caching. Default is 0. Tests have shown that this cache only improves search time with boolean queries, wildcard queries and fuzzy queries. For simple queries like a term query or a match all query this cache has a negative impact on performance

group.query  [query] Return a single group of documents that also match the given query.

group.format  One of grouped or simple. If simple, the grouped documents are presented in a single flat list. The start and rows parameters refer to numbers of documents instead of numbers of groups.

group.func  [function query] Group based on the unique values of a function query. <!> Solr4.0 This parameter only is supported on 4.0

base  URL endpoint.

callopts  Call options passed on to httr::GET

raw  (logical) If TRUE, returns raw data in format specified by wt param

parsetype  (character) One of ’list’ or ’df’

concat  (character) Character to concatenate elements of longer than length 1. Note that this only works reliably when data format is json (wt=’json’). The parsing is more complicated in XML format, but you can do that on your own.

verbose  If TRUE (default) the url call used printed to console.

...  Further args.

Value

XML, JSON, a list, or data.frame

References


See Also

solr_highlight, solr_facet
Examples

```r
## Not run:
url <- 'http://api.plos.org/search'

# Basic group query
solr_group(q='ecology', group.field='journal', group.limit=3, fl='id,score', base=url)
solr_group(q='ecology', group.field='journal', group.limit=3, fl='article_type', base=url)

# Different ways to sort (notice diff btw sort of group.sort)
# note that you can only sort on a field if you return that field
solr_group(q='ecology', group.field='journal', group.limit=3, fl=c('id','score'), base=url)
solr_group(q='ecology', group.field='journal', group.limit=3, fl=c('id','score','alm_twitterCount'),
          group.sort='alm_twitterCount desc', base=url)
solr_group(q='ecology', group.field='journal', group.limit=3, fl=c('id','score','alm_twitterCount'),
          sort='score asc', group.sort='alm_twitterCount desc', base=url)

# Two group.field values
out <- solr_group(q='ecology', group.field=c('journal','article_type'), group.limit=3, fl='id',
                 base=url, raw=TRUE)
solr_parse(out)
solr_parse(out, 'df')

# Get two groups, one with alm_twitterCount of 0-10, and another group with 10 to infinity
solr_group(q='ecology', group.limit=3, fl=c('id','alm_twitterCount'),
           group.query=c('alm_twitterCount:[0 TO 10]', 'alm_twitterCount:[10 TO *]'),
           base=url)

# Use of group.format and group.simple.
## The raw data structure of these two calls are slightly different, but
## the parsing inside the function outputs the same results. You can of course
## set raw=TRUE to get back what the data actually look like
solr_group(q='ecology', group.field='journal', group.limit=3, fl=c('id','score'),
           group.format='simple', base=url)
solr_group(q='ecology', group.field='journal', group.limit=3, fl=c('id','score'),
           group.format='grouped', base=url)
solr_group(q='ecology', group.field='journal', group.limit=3, fl=c('id','score'),
           group.format='grouped', group.main='true', base=url)

## End(Not run)
```

Description

Do highlighting searches, outputing highlight only.
Usage

```
solr_highlight(q, hl.fl = NULL, hl.snippets = NULL, hl fragsize = NULL, 
hl.q = NULL, hl.mergeContiguous = NULL, hl.requireFieldMatch = NULL, 
hl.maxAnalyzedChars = NULL, hl.alternateField = NULL, 
.hl.maxAlternateFieldLength = NULL, hl.preserveMulti = NULL, 
.hl.maxMultiValuedToExamine = NULL, hl.maxMultiValuedToMatch = NULL, 
.hl.formatter = NULL, hl.simple.pre = NULL, hl.simple.post = NULL, 
.hl.fragmenter = NULL, hl.fragListBuilder = NULL, 
.hl.fragmentsBuilder = NULL, hl.boundaryScanner = NULL, 
.hl.bs.maxScan = NULL, hl.bs.chars = NULL, hl.bs.type = NULL, 
.hl.bs.language = NULL, hl.bs.country = NULL, 
.hl.useFastVectorHighlighter = NULL, hl.usePhraseHighlighter = NULL, 
.hl.highlightMultiTerm = NULL, hl.regex.slop = NULL, 
.hl.regex.pattern = NULL, hl.regex.maxAnalyzedChars = NULL, start = 0, 
.rows = NULL, wt = "json", raw = FALSE, key = NULL, base = NULL, 
callopts = list(), fl = "DOES_NOT_EXIST", fq = NULL, 
parsetype = "list", verbose = TRUE)
```

Arguments

- **q**: Query terms. See examples.
- **hl.fl**: A comma-separated list of fields for which to generate highlighted snippets. If left blank, the fields highlighted for the LuceneQParser are the defaultSearchField (or the df param if used) and for the DisMax parser the qf fields are used. A '*' can be used to match field globs, e.g. 'text_*' or even '*' to highlight on all fields where highlighting is possible. When using '*', consider adding hl.requireFieldMatch=TRUE.
- **hl.snippets**: Max no. of highlighted snippets to generate per field. Note: it is possible for any number of snippets from zero to this value to be generated. This parameter accepts per-field overrides. Default: 1.
- **hl.fragsize**: The size, in characters, of the snippets (aka fragments) created by the highlighter. In the original Highlighter, "0" indicates that the whole field value should be used with no fragmenting. See http://wiki.apache.org/solr/HighlightingParameters for more info.
- **hl.q**: Set a query request to be highlighted. It overrides q parameter for highlighting. Solr query syntax is acceptable for this parameter.
- **hl.mergeContiguous**: Collapse contiguous fragments into a single fragment. "true" indicates contiguous fragments will be collapsed into single fragment. This parameter accepts per-field overrides. This parameter makes sense for the original Highlighter only. Default: FALSE.
- **hl.requireFieldMatch**: If TRUE, then a field will only be highlighted if the query matched in this particular field (normally, terms are highlighted in all requested fields regardless of which field matched the query). This only takes effect if "hl.usePhraseHighlighter" is TRUE. Default: FALSE.
hl.maxAnalyzedChars
How many characters into a document to look for suitable snippets. This parameter makes sense for the original Highlighter only. Default: 51200. You can assign a large value to this parameter and use hl fragsize=0 to return highlighting in large fields that have size greater than 51200 characters.

hl.alternateField
If a snippet cannot be generated (due to no terms matching), you can specify a field to use as the fallback. This parameter accepts per-field overrides.

hl.maxAlternateFieldLength
If hl.alternateField is specified, this parameter specifies the maximum number of characters of the field to return. Any value less than or equal to 0 means unlimited. Default: unlimited.

hl.preserveMulti
Preserve order of values in a multiValued list. Default: FALSE.

hl.maxMultiValuedToExamine
When highlighting a multiValued field, stop examining the individual entries after looking at this many of them. Will potentially return 0 snippets if this limit is reached before any snippets are found. If maxMultiValuedToMatch is also specified, whichever limit is hit first will terminate looking for more. Default: Integer.MAX VALUE

hl.maxMultiValuedToMatch
When highlighting a multiValued field, stop examining the individual entries after looking at this many matches are found. If maxMultiValuedToExamine is also specified, whichever limit is hit first will terminate looking for more. Default: Integer.MAX VALUE

hl.formatter
Specify a formatter for the highlight output. Currently the only legal value is "simple", which surrounds a highlighted term with a customizable pre- and post text snippet. This parameter accepts per-field overrides. This parameter makes sense for the original Highlighter only.

hl.simple.pre
The text which appears before and after a highlighted term when using the simple formatter. This parameter accepts per-field overrides. The default values are "<em>" and "</em>" This parameter makes sense for the original Highlighter only. Use hl.tag.pre and hl.tag.post for FastVectorHighlighter (see example under hl.fragmentsBuilder)

hl.simple.post
The text which appears before and after a highlighted term when using the simple formatter. This parameter accepts per-field overrides. The default values are "<em>" and "</em>" This parameter makes sense for the original Highlighter only. Use hl.tag.pre and hl.tag.post for FastVectorHighlighter (see example under hl.fragmentsBuilder)

hl.fragmenter
Specify a text snippet generator for highlighted text. The standard fragmenter is gap (which is so called because it creates fixed-sized fragments with gaps for multi-valued fields). Another option is regex, which tries to create fragments that "look like" a certain regular expression. This parameter accepts per-field overrides. Default: "gap"

hl.fragListBuilder
Specify the name of SolrFragListBuilder. This parameter makes sense for FastVectorHighlighter only. To create a fragSize=0 with the FastVectorHighlighter, use the SingleFragListBuilder. This field supports per-field overrides.
**hl.fragmentsBuilder**
Specify the name of SolrFragmentsBuilder. This parameter makes sense for FastVectorHighlighter only.

**hl.boundaryScanner**
Configures how the boundaries of fragments are determined. By default, boundaries will split at the character level, creating a fragment such as "uick brown fox jumps over the la". Valid entries are breakIterator or simple, with breakIterator being the most commonly used. This parameter makes sense for FastVectorHighlighter only.

**hl.bs.maxScan**
Specify the length of characters to be scanned by SimpleBoundaryScanner. Default: 10. This parameter makes sense for FastVectorHighlighter only.

**hl.bs.chars**
Specify the boundary characters, used by SimpleBoundaryScanner. This parameter makes sense for FastVectorHighlighter only.

**hl.bs.type**
Specify one of CHARACTER, WORD, SENTENCE and LINE, used by BreakIteratorBoundaryScanner. Default: WORD. This parameter makes sense for FastVectorHighlighter only.

**hl.bs.language**
Specify the language for Locale that is used by BreakIteratorBoundaryScanner. This parameter makes sense for FastVectorHighlighter only. Valid entries take the form of ISO 639-1 strings.

**hl.bs.country**
Specify the country for Locale that is used by BreakIteratorBoundaryScanner. This parameter makes sense for FastVectorHighlighter only. Valid entries take the form of ISO 3166-1 alpha-2 strings.

**hl.useFastVectorHighlighter**
Use FastVectorHighlighter. FastVectorHighlighter requires the field is termVectors=on, termPositions=on and termOffsets=on. This parameter accepts per-field overrides. Default: FALSE

**hl.usePhraseHighlighter**
Use SpanScorer to highlight phrase terms only when they appear within the query phrase in the document. Default: TRUE.

**hl.highlightMultiTerm**
If the SpanScorer is also being used, enables highlighting for range/wildcard/fuzzy/prefix queries. Default: FALSE. This parameter makes sense for the original Highlighter only.

**hl.regex.slop**
Factor by which the regex fragmenter can stray from the ideal fragment size (given by hl fragsize) to accommodate the regular expression. For instance, a slop of 0.2 with fragsize of 100 should yield fragments between 80 and 120 characters in length. It is usually good to provide a slightly smaller fragsize when using the regex fragmenter. Default: .6. This parameter makes sense for the original Highlighter only.

**hl.regex.pattern**
The regular expression for fragmenting. This could be used to extract sentences (see example solrconfig.xml) This parameter makes sense for the original Highlighter only.

**hl.regex.maxAnalyzedChars**
Only analyze this many characters from a field when using the regex fragmenter (after which, the fragmenter produces fixed-sized fragments). Applying a com-
plicated regex to a huge field is expensive. Default: 10000. This parameter
makes sense for the original Highlighter only.

start  Record to start at, default to beginning.
rows   Number of records to return.
wt     (character) Data format to return. One of xml or json (default).
raw    (logical) If TRUE (default) raw json or xml returned. If FALSE, parsed data
        returned.
key    API key, if needed.
base   URL endpoint
callopts Call options passed on to http::GET
f1     Fields to return
fq     Filter query, this does not affect the search, only what gets returned
parsetype One of list of df (data.frame)
verbose If TRUE (default) the url call used printed to console.

Value

XML, JSON, a list, or data.frame

References

See http://wiki.apache.org/solr/HighlightingParameters for more information on high-
lighting.

See Also

solr_search, solr_facet

Examples

## Not run:
url <- 'http://api.plos.org/search'
solr_highlight(q='alcohol', hl.fl = 'abstract', rows=10, base = url)
solr_highlight(q='alcohol', hl.fl = c('abstract','title'), rows=3, base = url)

# Raw data back
## json
solr_highlight(q='alcohol', hl.fl = 'abstract', rows=10, base = url,
               raw=TRUE)
## xml
solr_highlight(q='alcohol', hl.fl = 'abstract', rows=10, base = url,
               raw=TRUE, wt='xml')
## parse after getting data back
out <- solr_highlight(q='alcohol', hl.fl = c('abstract','title'), hl fragsize=30,
                       rows=10, base = url, raw=TRUE, wt='xml')
solr_parse(out, parsetype='df')

## End(Not run)
Description

Solr "more like this" search

Usage

solr_mlt(q = "*:*", fq = NULL, mlt.count = NULL, mlt.fl = NULL,
        mlt.mintf = NULL, mlt.mindf = NULL, mlt.minwl = NULL,
        mlt.maxwl = NULL, mlt.maxqt = NULL, mlt.maxntp = NULL,
        mlt.boost = NULL, mlt.qf = NULL, fl = NULL, wt = "json", start = 0,
        rows = NULL, key = NULL, base = NULL, callopts = list(),
        raw = FALSE, parsetype = "df", concat = ",", verbose = TRUE)

Arguments

q Query terms, defaults to "*:*", or everything.
fq Filter query, this does not affect the search, only what gets returned
mlt.count The number of similar documents to return for each result. Default is 5.
mlt.fl The fields to use for similarity. NOTE: if possible these should have a stored
        TermVector. DEFAULT_FIELD_NAMES = new String[] "contents"
mlt.mintf Minimum Term Frequency - the frequency below which terms will be ignored
        in the source doc. DEFAULT_MIN_TERM_FREQ = 2
mlt.mindf Minimum Document Frequency - the frequency at which words will be ignored
        which do not occur in at least this many docs. DEFAULT_MIN_DOC_FREQ = 5
mlt.minwl minimum word length below which words will be ignored. DEFAULT_MIN_WORD_LENGTH
        = 0
mlt.maxwl maximum word length above which words will be ignored. DEFAULT_MAX_WORD_LENGTH
        = 0
mlt.maxqt maximum number of query terms that will be included in any generated query.
        DEFAULT_MAX_QUERY_TERMS = 25
mlt.maxntp maximum number of tokens to parse in each example doc field that is not stored
        with TermVector support. DEFAULT_MAX_NUM_TOKENS_PARSED = 5000
mlt.boost [true/false] set if the query will be boosted by the interesting term relevance.
        DEFAULT_BOOST = false
mlt.qf Query fields and their boosts using the same format as that used in DisMaxQ-
        ParserPlugin. These fields must also be specified in mlt.fl.
fl Fields to return. We force ‘id’ to be returned so that there is a unique identifier
        with each record.
wt Data type returned, defaults to ‘json’
start  Record to start at, default to beginning.
rows  Number of records to return. Defaults to 10.
key   API key, if needed.
base  URL endpoint.
callopts  Call options passed on to \texttt{httr::GET}
raw  (logical) If TRUE, returns raw data in format specified by wt param
parsetype  (character) One of 'list' or 'df'
concat  (character) Character to concatenate elements of longer than length 1. Note that this only works reliably when data format is json (wt='json'). The parsing is more complicated in XML format, but you can do that on your own.
verbose  If TRUE (default) the url call used printed to console.

Value

XML, JSON, a list, or data.frame

References

See \url{http://wiki.apache.org/solr/MoreLikeThis} for more information.

Examples

```r
## Not run:
url <- 'http://api.plos.org/search'
solr_mlt(q=':*:*', mlt.count=2, mlt.fl='abstract', fl='score', base=url,
         fq="doc_type:full")
solr_mlt(q=':*:*', rows=2, mlt.fl='title', mlt.mindf=1, mlt.mintf=1, fl='alm_twitterCount',
         base=url)
solr_mlt(q='title:"ecology" AND body:"cell"', mlt.fl='title', mlt.mindf=1, mlt.mintf=1,
         fl='counter_total_all', rows=5, base=url)
solr_mlt(q='ecology', mlt.fl='abstract', fl='title', rows=5, base=url)
solr_mlt(q='ecology', mlt.fl='abstract', fl=c('score','eissn'), rows=5, base=url)
solr_mlt(q='ecology', mlt.fl='abstract', fl=c('score','eissn'), rows=5, base=url)

# get raw data, and parse later if needed
out=solr_mlt(q='ecology', mlt.fl='abstract', fl='title', rows=2, base=url,
            raw=TRUE)
library(rjson)
fromJSON(out)
solr_parse(out, "df")
```

## End(Not run)
solr_parse

Parse raw data from solr_search, solr_facet, or solr_highlight.

Description

See details.

Usage

solr_parse(input, parsetype, concat)

## S3 method for class 'sr_facet'
solr_parse(input, parsetype = NULL, concat = ",")

## S3 method for class 'sr_high'
solr_parse(input, parsetype = "list", concat = ",")

## S3 method for class 'sr_search'
solr_parse(input, parsetype = "list", concat = ",")

## S3 method for class 'sr_mlt'
solr_parse(input, parsetype = "list", concat = ",")

## S3 method for class 'sr_stats'
solr_parse(input, parsetype = "list", concat = ",")

## S3 method for class 'sr_group'
solr_parse(input, parsetype = "list", concat = ",")

Arguments

input Output from solr_facet
parsetype One of 'list' or 'df' (data.frame)
concat Character to concatenate strings by, e.g., ':;' (character). Used in solr_parse.sr_search only.

Details

This is the parser used internally in solr_facet, but if you output raw data from solr_facet using raw=TRUE, then you can use this function to parse that data (a sr_facet S3 object) after the fact to a list of data.frame's for easier consumption. The data format type is detected from the attribute "wt" on the sr_facet object.
Solr search.

**Description**

Solr search.

**Usage**

```java
solr_search(q = "*:*", sort = NULL, start = 0, rows = NULL,
pageDoc = NULL, pageScore = NULL, fq = NULL, f1 = NULL,
defType = NULL, timeAllowed = NULL, qt = NULL, wt = "json",
NOW = NULL, TZ = NULL, echoHandler = NULL, echoParams = NULL,
key = NULL, base = NULL, callopts = list(), raw = FALSE,
parsetype = "df", concat = ",", ..., verbose = TRUE)
```

**Arguments**

- **q**: Query terms, defaults to "*:*", or everything.
- **sort**: Field to sort on. You can specify ascending (e.g., score desc) or descending (e.g., score asc), sort by two fields (e.g., score desc, price asc), or sort by a function (e.g., sum(x_f, y_f) desc, which sorts by the sum of x_f and y_f in a descending order).
- **start**: Record to start at, default to beginning.
- **rows**: Number of records to return. Defaults to 10.
- **pageDoc**: If you expect to be paging deeply into the results (say beyond page 10, assuming rows=10) and you are sorting by score, you may wish to add the pageDoc and pageScore parameters to your request. These two parameters tell Solr (and Lucene) what the last result (Lucene internal docid and score) of the previous page was, so that when scoring the query for the next set of pages, it can ignore any results that occur higher than that item. To get the Lucene internal doc id, you will need to add [docid] to the &fl list. e.g., `q=*:*&start=10&pageDoc=5&pageScore=1.345&fl=[docid]`.
- **pageScore**: See pageDoc notes.
- **fq**: Filter query, this does not affect the search, only what gets returned.
- **f1**: Fields to return.
- **defType**: Specify the query parser to use with this request.
- **timeAllowed**: The time allowed for a search to finish. This value only applies to the search and not to requests in general. Time is in milliseconds. Values <= 0 mean no time restriction. Partial results may be returned (if there are any).
- **qt**: Which query handler used.
- **wt**: Data type returned, defaults to 'json'.
- **NOW**: Set a fixed time for evaluating Date based expressions.
- **TZ**: Time zone, you can override the default.
### echoHandler
If the echoHandler parameter is true, Solr places the name of the handle used in
the response to the client for debugging purposes.

### echoParams
The echoParams parameter tells Solr what kinds of Request parameters should
be included in the response for debugging purposes, legal values include:
- none - don’t include any request parameters for debugging
- explicit - include the parameters explicitly specified by the client in the
  request
- all - include all parameters involved in this request, either specified explicitly
  by the client, or implicit because of the request handler configuration.

### key
API key, if needed.

### base
URL endpoint.

### callopts
Call options passed on to http::GET

### raw
(logical) If TRUE, returns raw data in format specified by wt param

### parsetype
(character) One of 'list' or 'df'

### concat
(character) Character to concatenate elements of longer than length 1. Note that
this only works reliably when data format is json (wt='json'). The parsing is
more complicated in XML format, but you can do that on your own.

### verbose
If TRUE (default) the url call used printed to console.

#### Value
- XML
- JSON
- a list
- data.frame

#### References

#### See Also
- `solr_highlight`
- `solr_facet`

#### Examples
```r
# Not run:
url <- 'http://api.plos.org/search'
solr_search(q='*:*', rows=2, fl='id', base=url)

# Search for word ecology in title and cell in the body
solr_search(q='title:ecology AND body:cell', fl='title', rows=5, base=url)

# Search for word "cell" and not "body" in the title field
solr_search(q='title:"cell" -title:"body", fl='title', rows=5, base=url)

# Wildcards
## Search for word that starts with "cell" in the title field
```
Proximity searching

Search for words "sports" and "alcohol" within four words of each other

Range searches

Search for articles with Twitter count between 5 and 10

Boosts

Assign higher boost to title matches than to body matches (compare the two calls)

Parse data, using the USGS BISON API

You can choose how to combine elements longer than length 1

Using the USGS BISON API (http://bison.usgs.orl.gov/services.html#solr)

FunctionQuery queries

Here, we search on the product of counter_total_all and alm_twitterCount

Here, search on the product of counter_total_all and alm_twitterCount, using

Papers with most citations

Using the USGS BISON API (http://bison.usgs.orl.gov/services.html#solr)

Here, we search on the product of counter_total_all and alm_twitterCount
solr_stats

```r
grows=5, fl=c('id', 'counter_total_all'), fq='doc_type:full', base=url)

## papers with most tweets
solr_search(q='^val:\"max(alm_twitterCount)\"',
  rows=5, fl=c('id', 'alm_twitterCount'),fq='doc_type:full', base=url)

## End(Not run)
```

Description

Get Solr stats.

Usage

```r
solr_stats(q = "*:*", stats.field = NULL, stats.facet = NULL,
  wt = "json", start = 0, rows = 0, key = NULL, base = NULL,
  callopts = list(), raw = FALSE, parsetype = "df", verbose = TRUE)
```

Arguments

- `q` Query terms, defaults to "*:*", or everything.
- `stats.field` The number of similar documents to return for each result.
- `stats.facet` You can not facet on multi-valued fields.
- `wt` Data type returned, defaults to 'json'.
- `start` Record to start at, default to beginning.
- `rows` Number of records to return. Defaults to 10.
- `key` API key, if needed.
- `base` URL endpoint.
- `callopts` Call options passed on to httr::GET.
- `raw` (logical) If TRUE, returns raw data in format specified by `wt` param.
- `parsetype` (character) One of 'list' or 'df'.
- `verbose` If TRUE (default) the url call used printed to console.

Value

XML, JSON, a list, or data.frame

References

See Also

solr_highlight, solr_facet, solr_search, solr_mlt

Examples

```r
## Not run:
url <- 'http://api.plos.org/search'
solr_stats(q='science', stats.field='counter_total_all', base=url, raw=TRUE)
solr_stats(q='title:ecology' AND body:cell',
    stats.field=c('counter_total_all','alm_twitterCount'), base=url)
solr_stats(q='ecology', stats.field=c('counter_total_all','alm_twitterCount'),
    stats.facet='journal', base=url)
solr_stats(q='ecology', stats.field=c('counter_total_all','alm_twitterCount'),
    stats.facet=c('journal', 'volume'), base=url)

# Get raw data, then parse later if you feel like it
## json
out <- solr_stats(q='ecology', stats.field=c('counter_total_all','alm_twitterCount'),
    stats.facet=c('journal', 'volume'), base=url, raw=TRUE)
library(json)
fromJSON(out)
solr_parse(out) # list
solr_parse(out, 'df') # data.frame

## xml
out <- solr_stats(q='ecology', stats.field=c('counter_total_all','alm_twitterCount'),
    stats.facet=c('journal', 'volume'), base=url, raw=TRUE, wt="xml")
library(XML)
xmlParse(out)
solr_parse(out) # list
solr_parse(out, 'df') # data.frame

# Get verbose http call information
library(httr)
solr_stats(q='ecology', stats.field='alm_twitterCount', base=url,
    callopts=verbose())

## End(Not run)
```
Index

*Topic package
  solr-package, 2

collectargs, 2

is_sr_facet, 3
is_sr_high(is_sr_facet), 3
is_sr_search(is_sr_facet), 3

makemultiargs, 3

solr(solr-package), 2
solr-package, 2
solr_all, 4
solr_facet, 5, 6, 13, 18, 23, 26
solr_group, 12
solr_highlight, 5, 11, 13, 14, 23, 26
solr_mlt, 19, 26
solr_parse, 11, 21
solr_search, 11, 18, 22, 26
solr_stats, 25