solrq Documentation

Release 1.1.1

Michał Jaworski

Mar 27, 2017

Contents

CHAPTER 1

solrq

`solrq` is a Python Solr query utility. It helps making query strings for Solr and also helps with escaping reserved characters. `solrq` is has no external dependencies and is compatibile with `python2.6`, `python2.7`, `python3.3`, `python3.4`, `python3.5`, `pypy` and `pypy3`. It might be compatibile with other python releases/implentations but this has not been tested yet or is no longer tested (e.g `python3.2`).

```
pip install solrq
```

And you're ready to go!

CHAPTER 2

usage

Everything in `solrq` is about `Q()` object. Drop into python repl and just feed it with bunch of field and search terms to see how it works:

```
>>> from solrq import Q
>>> # note: all terms in single Q object are implicitely joined with 'AND'
>>> query = Q(type="animal", species="dog")
>>> query
<Q: type:animal AND species:dog>

>>> # ohh, forgot about cats?
>>> query | Q(type="animal", species="cat")
<Q: (type:animal AND species:dog) OR (type:animal AND species:cat)>

>>># more a cat lover? Let's give them a boost boost
>>> Q(type="animal") & (Q(species="cat")^2 | Q(species="dog"))
<Q: type:animal AND ((species:cat^2) OR species:dog)>
```

But what to do with this `Q`? Simply pass it to your Solr library of choice, like pysolr or mysolr. Most of python Solr libraries expect simple string as a query parameter and do not bother with escaping of reserved characters so you must take care of that by yourself. This is why `solrq` integrates so easily. Here is an example how you can use it with pysolr:

```
from solrq import Q
import pysolr

solr = Solr("<your solr url>")

# simply using Q object
solr.search(Q(text="easy as f***"))

# or explicitely making it string
solr.search(str(Q(text="easy as f***")))
```

quick reference

Full reference can be found in API reference documentation page but here is a short reference.

boosting queries

Use python ^ operator:

```
>>> Q(text='cat') ^ 2
<Q: text:cat^2>
```

AND queries

Use python & operator:

```
>>> Q(text='cat') & Q(text='dog')
<Q: text:cat AND text:dog>
```

OR queries

Use python | operator:

```
>>> Q(text='cat') | Q(text='dog')
<Q: text:cat OR text:dog>
```

NOT queries

Use python ~ operator:

```
>>> ~ Q(text='cat')
<Q: !text:cat>
```

ranges

Use `solrq.Range` wrapper:

```
>>> from solrq import Range
>>> Q(age=Range(18, 25))
<Q: age:[18 TO 25]>
```

proximity searches

Use `solrq.Proximity` wrapper:

```
>>> from solrq import Proximity
>>> Q(age=Proximity("cat dogs", 5))
<Q: age:"cat\ dogs"~5>
```

safe strings

All raw string values are treated as unsafe by default and will be escaped to ensure that final query string will not be broken by some rougue search value. This of course can be disabled if you know what you're doing using `Value` wrapper:

```
>>> from solrq import Q, Value
>>> Q(type='foo bar[]')
<Q: type:foo\ bar\[\]>
>>> Q(type=Value('foo bar[]', safe=True))
<Q: type:foo bar[]>
```

timedeltas, datetimes

Simply as:

```
>>> from datetime import datetime, timedelta
>>> Q(date=datetime(1970, 1, 1))
<Q: date:"1970-01-01T00:00:00Z">
>>> # note that timedeltas has any sense mostly with ranges
>>> Q(delta=timedelta(days=1))
<Q: delta:NOW+1DAYS+0SECONDS+0MILLISECONDS>
```

field wildcard

If you need to use wildcards in field names just use dict and unpack it inside of `Q()` instead of using keyword arguments:

```
>>> Q(**{"*_t": "text_to_search"})
<Q: *_t:text_to_search>
```

CHAPTER 3

contributing

Any contribution is welcome. Issues, suggestions, pull requests - whatever. There are no strict contribution guidelines beyond PEP-8 and sanity. Code style is checked with flakes8 and any PR that has failed build will not be merged.

One thing: if you submit a PR please do not rebase it later unless you are asked for that explicitely. Reviewing pull requests that suddenly had their history rewritten just drives me crazy.

CHAPTER 4

testing

Tests are run using tox. Simply install it and run:

```
pip install tox
tox
```

And that's all.

CHAPTER 5

Detailed documentation

API reference

class solrq.**Proximity**(*raw*, *distance*, *safe=False*)
Bases: *solrq.Value*

Wrapper around proximity value searches.

Parameters

- **raw** (*str*) – string of words for proximity search.
- **distance** (*int*) – distance between words.

Examples

```
>>> Proximity('foo bar', 4)
<Proximity: "foo\ bar"~4>
>>> Proximity('foo bar', 4, True)
<Proximity: "foo bar"~4>
```

Note: *Proximity* will in fact accept any type as a raw value that has __str__ method defined so it is developer's responsibility to make sure that raw has a reasonable value.

class solrq.**Q**(*children=None*, *op=<bound method type.and_ of <class 'solrq.QOperator'>>*, ***kwargs*)
Bases: object

Class for handling Solr queries in a semantic way.

Parameters

- **children** (*iterable*) – iterable of children Q objects. **Note**: can't be used with kwargs.
- **op** (*callable*) – operator to join query parts.

- **kwargs** (*dict*) – list of query parts. Note: can't be used with children.

Examples

```
>>> Q(foo="bar")
<Q: foo:bar>
>>> str(Q(foo="bar"))
'foo:bar'
```

```
>>> Q(text="Skyrim")
<Q: text:Skyrim>
```

```
>>> Q(language="EN", text="Skyrim")
<Q: ...>
```

```
>>> ~(Q(language="EN", text="cat") | Q(language="PL", text="dog"))
<Q: !((... AND ...) OR (... AND ...))>
```

Note: it is possible to specify query params that are not valid python argument names using dictionary unpacking e.g.:

```
>>> Q(**{"*_t": "text_to_search"})
<Q: *_t:text_to_search>
```

---

compile (*extra_parenthesis=False*)
Compile *Q* object into query string.

Parameters **extra_parenthesis** (*bool*) – add extra parenthesis to children query.

Returns compiled query string.

Return type str

Examples

```
>>> (Q(type="animal") & Q(name="cat")).compile()
'type:animal AND name:cat'
>>> (Q(type="animal") & Q(name="cat")).compile(True)
'(type:animal AND name:cat)'
```

class solrq.**QOperator**
Bases: object

Simply a namespace for handling Q object operator routines.

classmethod and_ (*qs_list*)
Perform 'and' operator routine.

Parameters **qs_list** (*iterable*) – iterable of "compiled" query strings.

Returns query strings joined with Solr *AND* operator as single string.

Return type str

classmethod **boost** (*qs_list*, *factor*)
Perform 'boost' operator routine.

Parameters

- **qs_list** (*iterable*) – single element list with compiled query string
- **factor** (*float or int*) – boost factor

Returns compiled query string followed with '~' and boost factor

Return type str

Note: this operator routine is not intended to be directly used as *Q* object argument but rather as a component for actual operator e.g:

```
>>> from functools import partial
>>> Q(children=[Q(a='b')], op=partial(QOperator.boost, factor=2))
<Q: a:b^2>
```

---

classmethod **not_** (*qs_list*)
Perform 'not' operator routine.

Parameters **qs_list** (*iterable*) – single item iterable of "compiled" query strings.

Returns

string with containing Solr ! operator followed by query. string.

Return type str

Note: `qs_list` must be a list despite 'not' operator accepts only single query string here, to avoid more complexity in *Q* objects initialization.

---

classmethod **or_** (*qs_list*)
Perform 'or' operator routine.

Parameters **qs_list** (*iterable*) – iterable of "compiled" query strings.

Returns query strings joined with Solr *OR* operator as single string.

Return type str

class `solrq.`**Range** (*from_*, *to*, *safe=None*, *boundaries='inclusive'*)
Bases: *solrq.Value*

Wrapper around range values.

Wraps two values with Solr's `[<from> TO <to>]` syntax (defaults to inclusive boundaries) with respect to restricted character escaping.

Wraps two values with Solr's `[<from> TO <to>]` (defaults to inclusive boundaries) syntax with respect to restricted character escaping.

Parameters

- **from** (*object*) – start of range, same as parameter `raw` in *Value*.
- **to** (*object*) – end of range, same as parameter `raw` in *Value*.

- **boundaries** (*str*) – type of boundaries for the range. Defaults to `'inclusive'`. Allowed values are:
  - **inclusive, ii, or []: translates to** `[<from> TO <to>]`
  - **exclusive, ee, or {}: translates to** `{<from> TO <to>}`
  - `ei`, or `{]`: translates to `{<from> TO <to>]`
  - `ie`, or `[}`: translates to `[<from> TO <to>}`

Examples

Simpliest range that matches all documents with some field set:

```
>>> Range('*', '*', safe=True)
<Range: [* TO *]>
```

Note that there are shortucts already provided:

```
>>> Range(ANY, ANY)
<Range: [* TO *]>
>>> SET
<Range: [* TO *]>
```

Other data types:

```
>>> Range(0, 20)
<Range: [0 TO 20]>
>>> Range(timedelta(days=2), timedelta())
<Range: [NOW+2DAYS+0SECONDS+0MILLISECONDS TO NOW]>
```

To use exclusive or mixed boundaries use `boundaries` argument:

```
>>> Range(0, 20, boundaries='exclusive')
<Range: {0 TO 20}>
>>> Range(0, 20, boundaries='ei')
<Range: {0 TO 20]>
>>> Range(0, 20, boundaries='[}')
<Range: [0 TO 20}>
```

Note: We could treat any iterables always as ranges when initializing *Q* objects but "explicit is better than implicit" and also this would require to handle value representation there and we don't want to do that.

BOUNDARY_BRACKETS = {'exclusive': '{}', 'ei': '{]', 'inclusive': '[]', 'ee': '{}', '{]': '{]', '[]': '[]', 'ii': '[]', '[}': '[}', '{}': '{

class solrq.**Value**(*raw*, *safe=False*)

Bases: object

Wrapper around query values.

It allows easy handling of character escaping and further query value translations.

By default it escapes all restricted characters so query can't be easily broken with unsafe strings. Also it recognizes `timedelta` and `datetime` objects so they can be represented in format that Solr can recognize (useful with ranges, see: *Range*)

Parameters

- **raw** (*object*) – raw value object. Must be string, datetime, timedelta or have __str__ method defined.
- **safe** (*bool*) – set to True to turn off character escaping.

Examples

In most cases you will pass string:

```
>>> Value("foo bar")
<Value: foo\ bar>
```

But it can be anything that has __str__ method:

```
>>> Value(1)
<Value: 1>
>>> Value(timedelta(days=1))
<Value: NOW+1DAYS+0SECONDS+0MILLISECONDS>
>>> Value(Value("foo"))
<Value: foo>
```

To get final query string just make it str:

```
>>> str(Value("foo bar"))
'foo\\ bar'
```

Note that raw strings are not safe by default:

```
>>> Value('foo [] bar')
<Value: foo\ \[\]\ bar>
>>> Value("foo [] bar", safe=True)
<Value: foo [] bar>
```

ESCAPE_RE = <_sre.SRE_Pattern object>

TIMEDELTA_FORMAT = 'NOW{days:+d}DAYS{secs:+d}SECONDS{mills:+d}MILLISECONDS'

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index