mysolr Documentation

Release 0.8.2

Rubén Abad, Miguel Olivares

June 05, 2014

Contents

mysolr was born to be a fast and easy-to-use client for Apache Solr's API and because existing Python clients didn't fulfill these conditions.

Since version 0.5 mysolr supports Python 3 except concurrent search feature.

CHAPTER 1

Basic Usage

```
from mysolr import Solr

# Default connection to localhost:8080
solr = Solr()

# All solr params are supported!
query = {'q' : '*:*', 'facet' : 'true', 'facet.field' : 'foo'}
response = solr.search(**query)

# do stuff with documents
for document in response.documents:
    # modify field 'foo'
    document['foo'] = 'bar'

# update index with modified documents
solr.update(response.documents, commit=True)
```

CHAPTER 2

Contents

2.1 Installation

To install mysolr from Pypi:

```
pip install mysolr
```

From source code:

```
python setup.py install
```

2.1.1 Dependencies

requests

Mysolr uses requests module for sending HTTP requests. So, if you install mysolr from source code you have to install it.

anyjson

New in version 0.8.1.

Since using eval could be dangerous. We decided to use anyjson which tries to use the fastest json library in your enviroment, using simplejson by default.

2.1.2 Concurrent search

Concurrent search feature is only available for python 2.X because it depends on Gevent and grequests. So if you want to use this feature, you have to install it as an extra.

```
pip install "mysolr[async]"
```

2.2 User Guide

2.2.1 Connecting to Solr

Use mysolr.Solr object to connect to a Solr instance.

```
from mysolr import Solr

# Default connection. Connecting to http://localhost:8080/solr/
solr = Solr()

# Custom connection
solr = Solr('http://foo.bar:9090/solr/')
```

New in version 0.9.

You can reuse HTTP connection by using requests.Session object

```
from mysolr import Solr
import requests

session = requests.Session()
solr = Solr('http://localhost:8983/solr/collection1', make_request=session)
```

New in version 0.9.

Using a requests.Session object allows you to connect to servers secured with HTTP basic authentication as follows:

```
from mysolr import Solr
import requests

session = requests.Session()
session.auth = ('admin', 'admin')
solr = Solr('http://localhost:8983/solr/collection1', make_request=session)
```

New in version 0.8.

Solr 4.0 changed a bit the api so, Solr object will guess the solr server version by making a request. You can manually set the solr version with the paremeter *version*

```
from mysolr import Solr

# Default connection. Connecting to a solr 4.X server
solr = Solr(version=4)
```

2.2.2 Queriying to Solr

Making a query to Solr is very easy, just call search method with your query.

```
from mysolr import Solr

solr = Solr()
# Search for all documents
response = solr.search(q='*:*')
# Get documents
documents = response.documents
```

Besides, all available Solr query params are supported. So making a query using pagination would be as simple as

```
from mysolr import Solr

solr = Solr()

# Get 10 documents
response = solr.search(q='*:*', rows=10, start=0)
```

Some parameters contain a period. In those cases you have to use a dictionary to build the query:

```
from mysolr import Solr

solr = Solr()

query = {'q' : '*:*', 'facet' : 'true', 'facet.field' : 'foo'}
response = solr.search(**query)
```

Sometimes specifying a HTTP parameter multiple times is needed. For instance when faceting by several fields. Use a list in that case.:

```
from mysolr import Solr

solr = Solr()

query = {'q' : '*:*', 'facet' : 'true', 'facet.field' : ['foo', 'bar']}
response = solr.search(**query)
```

2.2.3 Cursors

The typical concept of cursor in relational databases is also implemented in mysolr.

```
from mysolr import Solr

solr = Solr()

cursor = solr.search_cursor(q='*:*')

# Get all the documents
for response in cursor.fetch(100):
    # Do stuff with the current 100 documents
    pass
```

2.2.4 Facets

This is a query example using facets with mysolr.

```
from mysolr import Solr

solr = Solr()
# Search for all documents facets by field foo
query = {'q' : '*:*', 'facet' : 'true', 'facet.field' : 'foo'}
response = solr.search(**query)
# Get documents
documents = response.documents
# Get facets
facets = response.facets
```

Facets are parsed and can be accessed by retrieving `facets` attribute from the SolrResponse object. Facets look like this:

```
{
    'facet_dates': {},
    'facet_fields': {'foo': OrderedDict[('value1', 2), ('value2', 2)]},
    'facet_queries': {},
    'facet_ranges': {}
}
```

Ordered dicts are used to store the facets because order matters.

In any case, if you don't like how facets are parsed you can use `raw_content` attribute which contains the raw response from solr.

2.2.5 Spellchecker

This is an example of a query that uses the spellcheck component.

```
from mysolr import Solr

solr = Solr()

# Spell check query
query = {
    'q' : 'helo wold',
    'spellcheck' : 'true',
    'spellcheck.collate': 'true',
    'spellcheck.build':'true'
}

response = solr.search(**query)
```

Spellchecker results are parsed and can be accessed by getting the `spellcheck` attribute from the SolrResponse object.:

```
{'collation': 'Hello world',
'correctlySpelled': False,
'suggestions': {
                'helo': {'endOffset': 4,
                            'numFound': 1,
                            'origFreq': 0,
                            'startOffset': 0,
                            'suggestion': [{'freq': 14,
                                            'word': 'hello'}]},
                'wold': {'endOffset': 9,
                        'numFound': 1,
                        'origFreq': 0,
                        'startOffset': 5,
                        'suggestion': [{'freq': 14, 'word': 'world'}]}}}
```

2.2.6 Stats

`stats` attribute is just a shortcut to stats result. It is not parsed and has the format sent by Solr.

2.2.7 Highlighting

Like stats, `highlighting` is just a shortcut.

2.2.8 Concurrent searchs

As mysolr is using requests, it is posible to make concurrent queries thanks to grequest

```
from mysolr import Solr
solr = Solr()
# queries
queries = [
    {
        'q' : '*:*'
    },
    {
        'q' : 'foo:bar'
    }
]

# using 10 threads
responses = solr.async_search(queries, size=10)
```

See *installation* section for further information about how to install this feature.

2.2.9 Indexing documents

```
from mysolr import Solr

solr = Solr()

# Create documents
documents = [
    {'id' : 1,
     'field1' : 'foo'
    },
    {'id' : 2,
     'field2' : 'bar'
    }
]
# Index using json is faster!
solr.update(documents, 'json', commit=False)

# Manual commit
solr.commit()
```

2.3 Recipes

2.3.1 Solr backup

How to copy all documents from one solr server to another.

```
from mysolr import Solr

PACKET_SIZE = 5000

solr_source = Solr('http://server1:8080/solr/')
solr_target = Solr('http://server2:8080/solr/')
```

```
cursor = solr_source.search_cursor(q='*:*')

for resp in cursor.fetch(PACKET_SIZE):
    source_docs = resp.documents
    solr_target.update(source_docs)
```

2.4 Classes

2.4.1 Solr class

class `mysolr.Solr`(*base_url='http://localhost:8080/solr/'*, *make_request=<module 'requests' from '/home/docs/checkouts/readthedocs.org/user_builds/mysolr/envs/latest/local/lib/python2.7/site-packages/requests/__init__.pyc'>*, *use_get=False*, *version=None*)

Acts as an easy-to-use interface to Solr.

`async_search`(*queries*, *size=10*, *resource='select'*)

Asynchronous search using async module from requests.

Parameters

- **queries** – List of queries. Each query is a dictionary containing any of the available Solr query parameters described in http://wiki.apache.org/solr/CommonQueryParameters. 'q' is a mandatory parameter.
- **size** – Size of threadpool
- **resource** – Request dispatcher. 'select' by default.

`commit`(*wait_flush=True*, *wait_searcher=True*, *expunge_deletes=False*)

Sends a commit message to Solr.

Parameters

- **wait_flush** – Block until index changes are flushed to disk (default is True).
- **wait_searcher** – Block until a new searcher is opened and registered as the main query searcher, making the changes visible (default is True).
- **expunge_deletes** – Merge segments with deletes away (default is False)

`delete_by_key`(*identifier*, *commit=True*)

Sends an ID delete message to Solr.

Parameters **commit** – If True, sends a commit message after the operation is executed.

`delete_by_query`(*query*, *commit=True*)

Sends a query delete message to Solr.

Parameters **commit** – If True, sends a commit message after the operation is executed.

`get_system_info`()

Gets solr system status.

`is_up`()

Check if a Solr server is up using ping call

`more_like_this`(*resource='mlt'*, *text=None*, ***kwargs*)

Implements convenient access to Solr MoreLikeThis functionality

Please, visit http://wiki.apache.org/solr/MoreLikeThis to learn more about MLT configuration and common parameters.

There are two ways of using MLT in Solr:

Using a previously configured RequestHandler You normally specify a query and the first matching document for that query is used to retrieve similar documents. You can however specify a text instead of a query, and similar documents to the text will be returned. You must configure a MLT RequestHandler in your solrconfig.xml in order to get advantage of this functionality. Note that this method has a default resource name with value "mlt", but if your RequestHandler has a different name you must specify it when calling the more_like_this method.

Using the MLT Search Component: The resulting documents in this case will be those that match the regular query, but the SolrResponse will have a "mlt" section where similar documents for each result document will be given.

Parameters

- **resource** – Request dispatcher. 'ml' by default.
- **text** – Text to use for similar documents retrieval. None by default.
- ****kwargs** – Dictionary containing any of the available Solr query parameters described in http://wiki.apache.org/solr/CommonQueryParameters or MoreLikeThis Common parameters described in http://wiki.apache.org/solr/MoreLikeThis. 'q' is a mandatory parameter in all cases except when using a MLT RequestHandler with a Text parameter.

`optimize`(*wait_flush=True*, *wait_searcher=True*, *max_segments=1*)
Sends an optimize message to Solr.

Parameters

- **wait_flush** – Block until index changes are flushed to disk (default is True)
- **wait_searcher** – Block until a new searcher is opened and registered as the main query searcher, making the changes visible (default is True)
- **max_segments** – Optimizes down to at most this number of segments (default is 1)

`ping`()
Ping call to solr server.

`rollback`()
Sends a rollback message to Solr server.

`search`(*resource='select'*, ***kwargs*)
Queries Solr with the given kwargs and returns a SolrResponse object.

Parameters

- **resource** – Request dispatcher. 'select' by default.
- ****kwargs** – Dictionary containing any of the available Solr query parameters described in http://wiki.apache.org/solr/CommonQueryParameters. 'q' is a mandatory parameter.

`search_cursor`(*resource='select'*, ***kwargs*)

`update`(*documents*, *input_type='json'*, *commit=True*)
Sends an update/add message to add the array of hashes(documents) to Solr.

Parameters

- **documents** – A list of solr-compatible documents to index. You should use unicode strings for text/string fields.

- **input_type** – The format which documents are sent. Remember that json is not supported until version 3.
- **commit** – If True, sends a commit message after the operation is executed.

2.4.2 SolrResponse class

class `mysolr.`**`SolrResponse`**`(`*http_response=None*`)`

Parse solr response and make it accesible.

`extract_errmessage``()`

Tries to extract an error message from a SolrResponse body content.

Useful for error identification (e.g.: indexation errors)

`parse_content``()`

Tries to parse the raw content to know if its a structured results response or an unstructured HTML page (usually resulting from an error)

`parse_facets``(`*solr_facets*`)`

Parse facets.

`parse_spellcheck``(`*solr_suggestions*`)`

Parse spellcheck result into a more readable format.

2.4.3 Cursor class

class `mysolr.`**`Cursor`**`(`*url, query, make_request=<module 'requests' from '/home/docs/checkouts/readthedocs.org/user_builds/mysolr/envs/latest/local/lib/python2.7/site-packages/requests/__init__.pyc'>, use_get=False*`)`

Implements the concept of cursor in relational databases

`fetch``(`*rows=None*`)`

Generator method that grabs all the documents in bulk sets of 'rows' documents

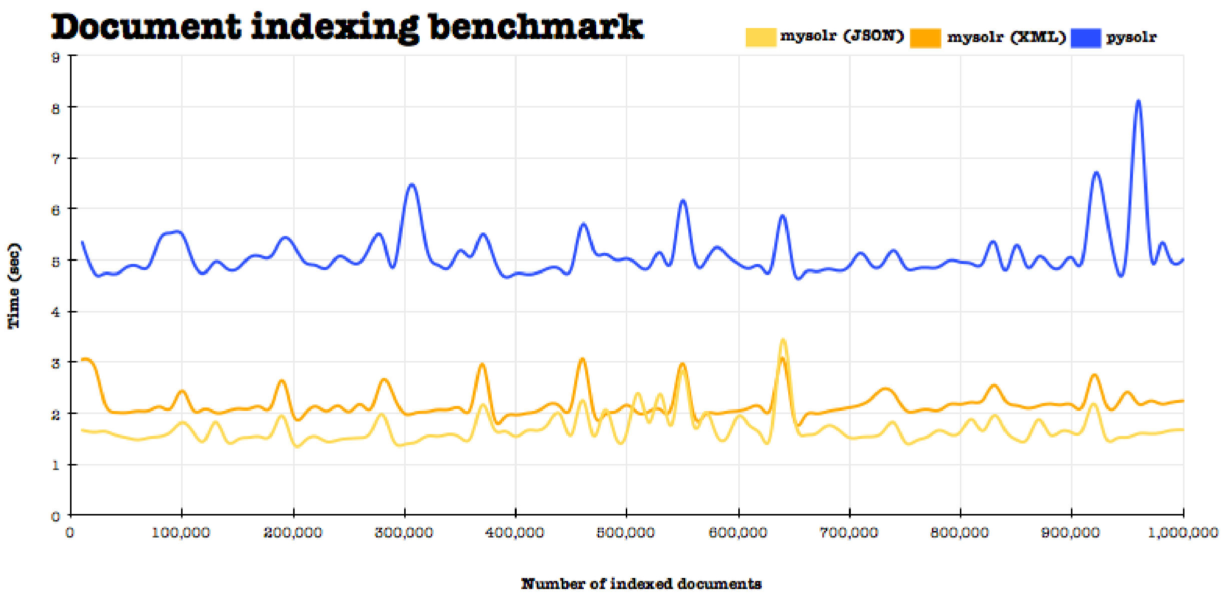
Parameters **rows** – number of rows for each request

2.5 Benchmark

One of the main goals of mysolr is to be the fastest python client of Solr. In this section you can see the performance of mysolr in different situations.

2.5.1 Indexing

The picture below is a comparison between mysolr and other clients at indexing time.


Document indexing benchmark
mysolr (JSON)
mysolr (XML)
pysolr
Time (sec)
0
1
2
3
4
5
6
7
8
9
0
100,000
200,000
300,000
400,000
500,000
600,000
700,000
800,000
900,000
1,000,000
Number of indexed documents

CHAPTER 3

References

We would like to thank the following developers their work and inspiration:

- The Apache Solr ‘s committers
- Kenneth Reitz, Requests creator

CHAPTER 4

Projects that are using mysolr

- solr_cli : Command line console for Apache Solr.

CHAPTER 5

Related projects

Other Python projects Apache Solr related:

- solrpy
- pysolr
- djangosolr

Python Module Index