wukong

Release 0.0.1

Oct 29, 2018

Contents

wukong offers an ORM query engine for Solr and Solr Cloud.

CHAPTER 1

Get Started

1.1 Define your document class

```
from wukong.models import SolrDoc

Class YourDocClass(SolrDoc):
    solr_hosts = "localhost:8080"
    collection_name = "my_solr_collection"
```

1.2 Fetch documents

```
docs = YourDocClass.documents.filter(firstname__eq="james").all()
```

1.3 Update documents

```
docs[0].firstname = "Jim"
docs[0].index() # single update

docs[1].firstname = "Smith"
docs.index() # batch update
```

1.4 Delete documents

```
docs[0].delete() # single delete
docs.delete() # batch delete
```

1.5 Create documents

```
YourDocClass.documents.create(id=1, firstname__eq="james", lastname__eq="bond")
```

CHAPTER 2

Documentations

2.1 Solr Document Class

In order to connect to your Solr collection, we can just extend a base class called *SolrDoc*. You can specify four attributes in your class.

- *solr_hosts*: the host name(s) for your Solr servers.
- *zookeeper_hosts*: the host name(s) for your Zookeeper hosts which monitor Solr. (optional)
- *collection_name*: the collection name for your collection in Solr.
- *request_timeout*: in how many seconds to drop the request to Solr. (optional, defaults to 15)

For example, if you have a collection named *User*, you can do the following.

```
from wukong.models import SolrDoc

Class User(SolrDoc):
    solr_hosts = "localhost:8080"
    zookeeper_hosts = "localhost:2181"
    collection_name = "users"

def validate_schema_fields(self, fields):
    pass

def get_data_for_solr(self):
    pass
```

You can overide existing methods to fit your business logic, like *validate_schema_fields*, *get_data_for_solr*.

- *validate_schema_fields*: return boolean to validate if the current document is consistent with the Solr Schema
- *get_data_for_solr*: return a json format to send to Solr for indexing

If you have multiple collections, you can define a base class to define *solr_hosts* and *zookeeper_hosts*, and the subclasses to only specify the *collection_name*.

```
from wukong.models import SolrDoc

Class BaseDoc(SolrDoc):
    solr_hosts = "localhost:8080"
    zookeeper_hosts = "localhost:2181"

Class User(BaseDoc)
    collection_name = "users"

Class Car(BaseDoc)
    collection_name = "cars"
```

2.2 Documents Retrieval

Once you define your document class, you can use it to fetch documents in Solr.

2.2.1 Filtering

```
# fetch all documents whose name is james
User.documents.filter(name__eq="james").all()

# fetch all documents whose name is not james
User.documents.filter(name__ne="james").all()

# fetch all documents whose name has james as substring
User.documents.filter(name__wc="james").all()

# fetch all documents whose name doesn't have james as substring
User.documents.filter(name__nwc="james").all()

# fetch all documents whose age is greater to 30
User.documents.filter(age__g=30).all()

# fetch all documents whose age is less to 30
User.documents.filter(age__l=30).all()

# fetch all documents whose age is greater or equal to 30
User.documents.filter(age__ge=30).all()

# fetch all documents whose age is less or equal to 30
User.documents.filter(age__le=30).all()

# fetch all documents who lives in either in Ottawa or New York
User.documents.filter(city__in=['Ottawa', 'New York']).all()

# fetch all documents who lives in neither in Ottawa nor New York
User.documents.filter(city__nin=['Ottawa', 'New York']).all()

# fetch all documents whose has zip field
User.documents.filter(zip__ex=True).all()

# fetch all documents whose doesn't have zip field
User.documents.filter(zip__nex=True).all()
```

(continues on next page)

(continued from previous page)

```
# fetch all documents whose age is less to 30 and live in Ottawa
User.documents.filter(age__l=30, city__eq="Ottawa").all()

# fetch all documents whose age is less to 30 or live in Ottawa
User.documents.filter(OR(age__l=30, city__eq="Ottawa")).all()

# fetch all documents whose age is less to 30 or live in Ottawa and also has zip field
User.documents.filter(AND(OR(age__l=30, city__eq="Ottawa"), zip__ex=True)).all()

# fetch all documents whose age is less to 30 and live in Ottawa
User.documents.filter(age__l=30).filter(city__eq="Ottawa").all()
```

2.2.2 Sorting

```
# fetch all documents sorted by age ascendingly
User.documents.sort_by('age').all()

# fetch all documents whose name is james sorted by age descendingly
User.documents.filter(name__eq="james").sort_by('-age').all()
```

2.2.3 Search

```
# fetch all documents matched `james bond` in the default field (usually `text`)
User.documents.search('james bond').all()

# fetch all documents matched `james bond` in name (weight 10) and city (weight 1)
User.documents.search('james bond', name=10, city=1).all()

# fetch all documents matched `james bond` in default field with at least 2 tokens␣
↪matched
User.documents.search('james bond', minimin_matches=2).all()
```

2.2.4 Grouping

```
# group all documents by `gender` and fetch the groups
User.documents.group_by('gender').groups()

# group all documents by `gender` and `city` and fetch the groups
User.documents.group_by(['gender', 'city']).groups()

# group all documents by `gender` and `city` and get 3 documents in each group
User.documents.group_by(['gender', 'city'], group_limit=3).groups()
```

2.2.5 Faceting

```
# facet all documents by `gender` and fetch the facets
User.documents.facet_by('gender').facets()
```

(continues on next page)

(continued from previous page)

```
# facet all documents by `gender` and `city` and fetch the facets
User.documents.facet_by(['gender', 'city']).facets()

# facet all documents by `gender` and `city` and fetch the facets at least having 10␣
↪docs
User.documents.facet_by(['gender', 'city'], mincount=10).groups()
```

2.2.6 Pagination

```
# paginate documents and get 100 documents starting from 200
User.documents.offset(200).limit(100).all()
```

2.2.7 Return Fields

```
# only fetch the fields (id and name) for each document
User.documents.only('id, 'name').all()
```

2.2.8 Raw Documents

```
# fetch all documents matched `james bond` and fetch a list of raw json rather than␣
↪SolrDoc list
User.documents.search('james bond').raw()
```

2.2.9 Chained Query

```
# fetch the documents matching `james bond` and with age greater than 30, and get 100␣
↪documents starting from 200
User.documents.search('james bond').filter(age__g=30).offset(200).limit(100).all()
```

2.3 Document Creation

```
# Create a document in Solr
User.documents.create(id=12345, name="James Bond", city="London")

# Batch create within one request to Solr
docs = [
    User(id=12345, name="James Bond", city="London"),
    Entity(id=12346, name="Kate", city="New York")
    ...
]
docs = SolrDocs(docs)
docs.index()
```

2.4 Document Update

```
doc = User.documents.create(id=12345, name="James Bond", city="London")

# Update a document in Solr
doc.name = "Jim Bond"
doc.city = "Ottawa"
doc.index()
```

2.5 Document Delete

```
doc = User.documents.create(id=12345, name="James Bond", city="London")

# Update a document in Solr
doc.delete()

# Batch delete within one request to Solr
docs = [
    User(id=12345, name="James Bond", city="London"),
    Entity(id=12346, name="Kate", city="New York")
    ...
]
docs = SolrDocs(docs)
docs.delete()
```

2.6 Complex Query

```
# You can always use `User.solr.select` to build your custom query
User.solr.select({
    q: "it is complex",
    ...,
    ...,
    ...
})
```

CHAPTER 3

Modules

- genindex
- modindex