lovely.esdb

Release 0.3.8

September 29, 2016

Contents

This package provides a simple elasticsearch document management. Its main purpose is to map ES documents to python classes with the possibility to work with raw ES data for simple JSON mappings.

CHAPTER 1

Features

- provide a `Document` class for ES documents
- allows property definition (currently untyped)
- `ObjectProperty` to be able to store any JSON pickle-able object
- automatic mapping of ES index data to `Document` classes
- manage different `Document` classes in the same index
- manage bulk operations for `Documents`
- `Document` proxy `LazyDocument` for lazy loading

CHAPTER 2

License

The MIT License (MIT) Copyright (c) 2016, Lovely Systems GmbH

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER 3

Contents

3.1 Usage

Contents

Here are some of the basics on how to use the `Document` class.

3.1.1 Implement a Document Class

Create your classes inherited from `Document`:

```
>>> from lovely.esdb.document import Document
>>> from lovely.esdb.properties import Property
>>> class MyDocument(Document):
...     INDEX = 'mydocindex'
...
...     id = Property(primary_key=True)
...     name = Property(default=u'')
...
...     def __repr__(self):
...         return '<%s [id=%r, name=%r]>' % (
...                     self.__class__.__name__,
...                     self.id,
...                     self.name,
...                 )
```

The class can now globally be connected to an elasticsearch host.

`lovely.esdb` uses the `elasticsearch` python package. We need an instance of the `Elasticsearch` class to connect our documents to a store:

```
>>> from elasticsearch import Elasticsearch
>>> es_client = Elasticsearch(['localhost:%s' % crate_port])
>>> MyDocument.ES = es_client
```

If you have only one elasticsearch cluster for your application it is also possible to set the ES client for all new classes globally:

```
>>> Document.ES = es_client
```

Create an index for the documents:

```
>>> es_client.indices.create(
...     index=MyDocument.INDEX,
...     body={
...         'settings': {'number_of_shards': 1},
...         "mappings" : {
...             "default" : {
...                 "properties" : {
...                     "id" : { "type" : "string", "index" : "not_analyzed" },
...                     "title" : { "type" : "string", "index" : "analyzed" }
...                 }
...             }
...         }
...     })
{u'acknowledged': True}
```

3.1.2 Create and Store Documents

That's all you need. Now you can use it:

```
>>> doc = MyDocument(id="1", name="John Doe")
>>> pprint(doc.store())
{u'_id': u'1',
 u'_index': u'mydocindex',
 u'_type': u'default',
 u'_version': 1,
 u'created': True}
```

3.1.3 Get a Document

To get a document back using its `primary key` use the `get` method of your class:

```
>>> MyDocument.get("1")
<MyDocument [id=u'1', name=u'John Doe']>
```

3.1.4 Get Multiple Documents

`mget` allows to get multiple documents by their `primary key`:

```
>>> pprint(MyDocument.mget(["1", "2"]))
[<MyDocument [id=u'1', name=u'John Doe']>,
 None]
```

3.1.5 Search Documents

`Document` provides a query method which allows to do any elasticsearch query. The difference is that the result hits are resolved as `Document` instances:

```
>>> _ = MyDocument.refresh()
>>> query = {
...     "query": {
...         "match": {
...             "name": "John Doe"
...         }
...     }
... }
>>> result = MyDocument.search(query)
>>> pprint(result)
{u'_shards': {u'failed': 0, u'successful': 1, u'total': 1},
 u'hits': {u'hits': [<MyDocument [id=u'1', name=u'John Doe']>],
           u'max_score': ...,
           u'total': 1},
 u'timed_out': False,
 u'took': ...}
>>> result['hits']['hits']
[<MyDocument [id=u'1', name=u'John Doe']>]
```

3.1.6 Delete a Document

Deleting a document is as easy as creating it:

```
>>> doc = MyDocument(id="2", name="to be deleted")
>>> _ = doc.store()
>>> pprint(doc.delete())
{u'_id': u'2',
 u'_index': u'mydocindex',
 u'_type': u'default',
 u'_version': 2,
 u'found': True}
```

3.2 Relations

Contents

3.2.1 1:1 Relation

A simple relation property allows to manage and resolve one to one relations between documents.

```
>>> from lovely.esdb.document import Document
>>> from lovely.esdb.properties import Property, LocalRelation
```

```
>>> class LocalDoc(Document):
...     """References RemoteDoc via the rel property.
...     """
...
...     INDEX = 'localdoc'
...
...     id = Property(primary_key=True)
...
...     # The relation is configured with the name/path to the local
...     # property on which the relation stores its internal data and the
...     # remote Document and property name. The remote property name must
...     # be the primary key of the remote Document.
...     rel = LocalRelation('ref.ref_id', 'RemoteDoc.id')
...
...     # ref is the property which is needed by the relation to store the
...     # local relation data.
...     ref = Property()
```

RemoteDoc is the referenced document. There is nothing special about it:

```
>>> class RemoteDoc(Document):
...     """Referenced document with only an id
...     """
...
...     INDEX = 'remotedoc'
...     ES = es_client
...
...     id = Property(primary_key=True)
...
...     def __repr__(self):
...         return "<RemoteDoc %r>" % self.id
```

Create an index on which the remote document can be stored:

```
>>> es_client.indices.create(
...     index=RemoteDoc.INDEX,
...     body={
...         'settings': {'number_of_shards': 1},
...         "mappings" : {
...             "default" : {
...                 "properties" : {
...                     "id" : { "type" : "string", "index" : "not_analyzed" },
...                 }
...             }
...         }
...     })
{u'acknowledged': True}
```

Create a document which can be used in LocalDoc:

```
>>> remote = RemoteDoc(id='1')
>>> _ = remote.store()

>>> local = LocalDoc()
>>> local.rel = remote
>>> local.rel()
<RemoteDoc u'1'>
```

The `ref` property contains the id of the referenced document:

```
>>> local.ref
{'ref_id': '1'}
```

It is also possible to assign the primary key to the relation property:

```
>>> remote2 = RemoteDoc(id='2')
>>> _ = remote2.store()

>>> local.rel = '2'
>>> local.rel()
<RemoteDoc u'2'>
```

3.2.2 1:n Relation

The simple 1:n relation maintains a local list with the ids of the related documents.

```
>>> from lovely.esdb.properties import LocalOne2NRelation
>>> class LocalOne2NDoc(Document):
...     """References RemoteDoc via the rel property.
...     """
...
...     INDEX = 'localone2ndoc'
...
...     id = Property(primary_key=True)
...
...     # The relation is configured with the name/path to the local
...     # property on which the relation stores its internal data and the
...     # remote Document and property name. The remote property name must
...     # be the primary key of the remote Document.
...     rel = LocalOne2NRelation('ref.ref_id', 'RemoteDoc.id')
...
...     # ref is the property which is needed by the relation to store the
...     # local relation data.
...     ref = Property()

>>> local = LocalOne2NDoc()

>>> local.rel = [remote]
```

The relation provides a resolver:

```
>>> local.rel
<ListRelationResolver RemoteDoc(['1'])>
```

The resolver allows access to the items:

```
>>> local.rel[0]
<ListItemRelationResolver[0] RemoteDoc[1]>

>>> local.rel[0]()
<RemoteDoc u'1'>
```

Item assignement:

```
>>> local.rel = [remote, '2', {'id': '3'}]
>>> local.rel
<ListRelationResolver RemoteDoc(['1', '2', '3'])>
```