mongomotor Documentation

Release 0.1

Juca Crispim

March 20, 2015

Contents

MongoMotor is a tiny integration of MongoEngine, a document-object mapper for python and mongodb, with Motor, an asynchronous driver for mongodb built on top of tornado's mainloop

Using MongoMotor you can define your documents as you already do with MongoEngine, use all query niceties you already know and do all db operations asynchronously using Motor.

CHAPTER 1

Installation

Straight-forward installation, using pip:

```
$ pip install mongomotor
```

And that's it!

CHAPTER 2

MongoMotor usage

To use MongoMotor is very similar to use of MongoEngine. To define your documents there's no difference, except on import. This is why we'll use the same example used on mongoengine's tutorial. We'll create a simple thumblelog.

2.1 Defining documents

To begin, lets define the following documents:

```
# In the imports here we change ``mongoengine`` to ``mongomotor``.
from mongomotor import connect, Document, EmbeddedDocument
from mongomotor.fields import (StringField, ReferenceField, ListField,
                               EmbeddedDocumentField)

# First creating the conection with database
connect('mongomotor-test')


# Here the documents are the same used in mongoengine's tutorial
class User(Document):
    email = StringField(required=True)
    first_name = StringField(max_length=50)
    last_name = StringField(max_length=50)


class Comment(EmbeddedDocument):
    content = StringField()
    name = StringField(max_length=120)


class Post(Document):
    title = StringField(max_length=120, required=True)
    author = ReferenceField(User)
    tags = ListField(StringField(max_length=30))
    comments = ListField(EmbeddedDocumentField(Comment))

    meta = {'allow_inheritance': True}


class TextPost(Post):
    content = StringField()
```

```
class ImagePost(Post):
    image_path = StringField()


class LinkPost(Post):
    link_url = StringField()
```

Now, the usage is practically the same of mongoengine. Lets see:

2.2 Adding data

To add a new document to database, we'll do everything as with mongoengine, but with the difference that when we use the save() method, we use the keyword `yield`

```
author = User(email='niceguy@example.com', first_name='Nice', last_name='Guy')
yield author.save()

post1 = TextPost(title='Fun with MongoMotor', author=author)
post1.content = 'Took a look at MongoEngine today, looks pretty cool.'
post1.tags = ['mongodb', 'motor', 'mongoengine', 'mongomotor']
yield post1.save()

post2 = LinkPost(title='MongoMotor Documentation', author=author)
post2.link_url = 'http://mongomotor-ptbr.readthedocs.org/pt/latest/'
post2.tags = ['mongomotor']
yield post2.save()
```

2.3 Accessing data

Now we already have some posts we can access them. Again, it's like with mongoengine, except we use `yield` when accessing database:

```
# Here listing all posts that inherited from Post
for post_future in Post.objects:
    post = yield post_future
    print(post.title)

# Here only TextPost from ``author``
for post_future in TextPost.objects.filter(author=author):
    post = yield post_future
    print(post.content)

# And here filtering by tags
for post_future in TextPost.objects(tags='mongomotor'):
    post = yield post_future
    print(post.content)

# We could use the method ``to_list()`` to transform a queryset
# into a list.
posts = yield TextPost.objects.filter(tags='mongomotor')[:10].to_list()
for post in posts:
    print(post.title)
```

Note: While it appears that each document is retrieved individually, in fact this is the same behavior of motor's

`fetch_index`, which, by its instance, retrieve the documents in large batches. Apeser de parecer que cada documento é recuperado individualmente (por causa deste monte de `yield`), na verdade é o

---

When we use `get()` we also need to use `yield`, like this:

```
post = yield TextPost.objects.get(title='Fun with MongoMotor')
```

The same to access a ReferenceField

```
author = yield post.author
```

to use the method `first()` which (obviously) returns the first result of the query

```
post = yield Post.objects.order_by('-title').first()
```

or when we delete some document from database:

```
yield post.delete()
```

We can use the aggregation methods too, like `sum()`, `count()`, `average()`...

```
total_posts = yield Post.objects.count()
tags_frequencies = yield Post.objects.item_frequencies('tags')
```

CHAPTER 3

Licence

MongoMotor is free software, licensed under the GPL version 3 or latter.

CHAPTER 4

Contributing

MongoMotor's code is hosted on gitlab and there is the issue tracker, too. Feel free to create a fork of the project, open issues, do merge requests...

CHAPTER 5

Documentation translations

Documentação do MongoMotor em português

Well, that's it! Thank you!