mongomotor Documentation

Versão 0.1

Juca Crispim

20 March, 2015

Conteúdo

O MongoMotor é uma pequena integração do MongoEngine , um document object mapper para python e mongodb, com o Motor, um driver assíncrono para mongodb feito usando o mainloop do tornado.

Usando o MongoMotor você pode definir seus documentos como você já faz com o MongoEngine, utilizar todas as facilidades para fazer as queries que você já conhece, e fazer as operações em banco de dados assincronamente utilizando o motor.

CAPÍTULO 1

Instalação

A instalação comum, via pip.

```
$ pip install mongomotor
```

E é isso!

CAPÍTULO 2

Uso do mongomotor

Usar o mongomotor é bem similar ao uso do mongoengine. Para definir seus documentos não há diferença, a não ser no import. Por isso, usaremos o mesmo exemplo usado no tutorial do mongoengine. Vamos criar um tumblelog simples.

2.1 Definindo documentos

Para começar, vamos definir os seguintes documentos:

```
# Os imports são como os do mongoengine, só alterando ``mongoengine``
# para ``mongomotor``.
from mongomotor import connect, Document, EmbeddedDocument
from mongomotor.fields import (StringField, ReferenceField, ListField,
                               EmbeddedDocumentField)
from tornado import gen

# Primeiro criando a conexão com o banco de dados.
connect('mongomotor-test')


# Aqui os documentos iguais aos do tutorial do mongoengine.
# Primeiro, definindo User. Intâncias de user serão os autores dos posts.
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()
```

Agora, o uso é praticamente igual ao do mongoengine. Vejamos:

2.2 Adicionando dados

Para adicionar um novo documento à base de dados, faremos tudo como no mongoengine, a diferença é que quando formos usar o método save, usaremos `yield`

```
author = User(email='ross@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', '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 Acessando os dados

Agora que já temos alguns posts, podemos acessá-los. Novamente é como o mongoengine, só com uns `yield` por aí. Vamos lá acessar os nossos dados:

```
# Aqui listando todos os posts que heraram de Post
for post_future in Post.objects:
    post = yield post_future
    print(post.title)

# Aqui só os TextPost do ator ``author``
for post_future in TextPost.objects.filter(author=author):
    post = yield post_future
    print(post.content)

# E aqui filtrando por tags
for post_future in TextPost.objects(tags='mongomotor'):
    post = yield post_future
    print(post.content)

# Poderiamos também usar o método ``to_list()`` para transformar
# um queryset em uma lista
posts = yield TextPost.objects.filter(tags='mongomotor')[:10].to_list()
```

```
for post in posts:
    print(post.title)
```

Nota: Apeser de parecer que cada documento é recuperado individualmente (por causa deste monte de `yield`), na verdade é o mesmo comportamento de `fetch_next` do mongomotor, que que por sua vez recupera os documentos em lotes grandes.

Quando usamos `get()` também precisamos usar `yield`, assim:

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

O mesmo quando vamos acessar um ReferenceField,

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

usar o método `first()` que (obviamente) retorna o primeiro resultado da query,

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

ou quando se vai apagar um documento do banco de dados:

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

A gente também pode usar os métodos de agregação do MongoEngine, como `sum()`, `count()`, `average()`...

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

CAPÍTULO 3

Licença

MongoMotor é software livre, licenciado sob a GPL versão 3 ou posterior.

CAPÍTULO 4

Contribuindo

O código do MongoMotor está hospedado no gitlab e por lá também está o issue tracker. Fique à vontade para criar um fork do projeto, abrir issues, fazer merge requests...

CAPÍTULO 5

Traduções da documentação

MongoMotor docs in english

Bom, é isso. Obrigado!