Mongo-Thingy

Refty

Apr 18, 2023

CONTENTS

***Mongo-Thingy* is the most idiomatic and friendly-yet-powerful way to use MongoDB with Python.**

It is an *"Object-Document Mapper"* that gives you full advantage of MongoDB schema-less design by **not** asking you to define schemas in your code.

What you'll get:

- a simple and robust pure-Python code base, with 100% coverage and few dependencies;
- PyMongo query language - no need to learn yet another one;
- both sync and async support! choose what suits you best;
- Thingy views - control what to show, and create fields based on other fields;
- swappable backend - wanna use SQLite behind the scenes? well, you can;
- versioning *(optional)* - rollback to any point in any thingy history;
- and more!

We support all Python and MongoDB versions supported by PyMongo, namely:

- CPython 3.7+ and PyPy3.7+
- MongoDB 3.6, 4.0, 4.2, 4.4, and 5.0.

As a backend, Mongo-Thingy supports the following libraries:

- Synchronous:
  - PyMongo (default)
  - Mongomock
  - MontyDB
- Asynchronous:
  - Motor (default when Motor is installed)
  - Motor with Tornado (default when Motor and Tornado are installed)
  - Mongomock-Motor

CHAPTER

ONE

INSTALL

```
pip install mongo-thingy
```

CHAPTER

TWO

EXAMPLES

2.1 First steps

2.1.1 Connect, insert and find thingies

```
>>> from mongo_thingy import connect, Thingy
>>> connect("mongodb://localhost/test")

>>> class User(Thingy):
...     pass

>>> user = User({"name": "Mr. Foo", "age": 42}).save()
>>> User.count_documents()
1
>>> User.find_one({"age": 42})
User({'_id': ObjectId(...), 'name': 'Mr. Foo', 'age': 42})
```

In an AsyncIO (or Tornado) environment, use the asynchronous class instead:

```
>>> from mongo_thingy import connect, AsyncThingy
>>> connect("mongodb://localhost/test")

>>> class User(AsyncThingy):
...     pass

>>> user = await User({"name": "Mr. Foo", "age": 42}).save()
>>> await User.count_documents()
1
>>> await User.find_one({"age": 42})
User({'_id': ObjectId(...), 'name': 'Mr. Foo', 'age': 42})
```

To use another backend than the default ones, just pass its client class with `client_cls`:

```
>>> import mongomock
>>> connect(client_cls=mongomock.MongoClient)
```

2.1.2 Update a thingy

```
>>> user.age
42
>>> user.age = 1337
>>> user.save()
User({'_id': ObjectId(...), 'name': 'Mr. Foo', 'age': 1337})
```

2.2 Thingy views power

2.2.1 Complete information with properties

```
>>> class User(Thingy):
...     @property
...     def username(self):
...         return "".join(char for char in self.name if char.isalpha())

>>> User.add_view(name="everything", defaults=True, include="username")
>>> user = User.find_one()
>>> user.view("everything")
{'_id': ObjectId(...), 'name': 'Mr. Foo', 'age': 1337, 'username': 'MrFoo'}
```

2.2.2 Hide sensitive stuff

```
>>> User.add_view(name="public", defaults=True, exclude="password")
>>> user.password = "t0ps3cr3t"
>>> user.view()
{'_id': ObjectId(...), 'name': 'Mr. Foo', 'age': 1337, 'password': 't0ps3cr3t'}
>>> user.view("public")
{'_id': ObjectId(...), 'name': 'Mr. Foo', 'age': 1337}
```

2.2.3 Only use certain fields/properties

```
>>> User.add_view(name="credentials", include=["username", "password"])
>>> user.view("credentials")
{'username': 'MrFoo', 'password': 't0ps3cr3t'}
```

2.2.4 Apply views on cursors

```
>>> cursor = User.find()
>>> for credentials in cursor.view("credentials"):
...     print(credentials)
{'username': 'MrFoo', 'password': 't0ps3cr3t'}
{'username': 'MrsBar', 'password': '123456789'}
...
```

And if your cursor is already exhausted, you can still apply a view!

```
>>> users = User.find().to_list(None)
>>> for credentials in users.view("credentials"):
...     print(credentials)
{'username': 'MrFoo', 'password': 't0ps3cr3t'}
{'username': 'MrsBar', 'password': '123456789'}
...
```

2.3 Versioning

```
>>> from mongo_thingy.versioned import Versioned

>>> class Article(Versioned, Thingy):
...     pass

>>> article = Article(content="Cogito ergo sum")
>>> article.version
0

>>> article.save()
Article({'_id': ObjectId('...'), 'content': 'Cogito ergo sum'})
>>> article.version
1

>>> article.content = "Sum ergo cogito"
>>> article.save()
Article({'_id': ObjectId('...'), 'content': 'Sum ergo cogito'})
>>> article.version
2

>>> article.revert()
Article({'_id': ObjectId('...'), 'content': 'Cogito ergo sum'})
```

(continues on next page)

(continued from previous page)

```
>>> article.version
3
```

2.4 Database/collection "discovery"

2.4.1 Default behaviour

```
>>> class AuthenticationGroup(Thingy):
...     pass

>>> connect("mongodb://localhost/")
>>> AuthenticationGroup.collection
Collection(Database(MongoClient(host=['localhost:27017'], ...), 'authentication'), 'group
↪')
```

2.4.2 Use mismatching names for Thingy class and database collection

You can either specify the collection name:

```
>>> class Foo(Thingy):
...   collection_name = "bar"
```

or the collection directly:

```
>>> class Foo(Thingy):
...   collection = db.bar
```

You can then check what collection is being used with:

```
>>> Foo.collection
Collection(Database(MongoClient('localhost', 27017), 'database'), 'bar')
```

2.5 Indexes

2.5.1 Create an index

```
>>> User.create_index("email", sparse=True, unique=True)
```

2.5.2 Add one or more indexes, create later

```
>>> User.add_index("email", sparse=True, unique=True)
>>> User.add_index("username")

>>> User.create_indexes()
```

2.5.3 Create all indexes of all thingies at once

```
>>> from mongo_thingy import create_indexes
>>> create_indexes()
```

2.6 Dealing with camelCase data

```
>>> from mongo_thingy.camelcase import CamelCase

>>> class SystemUser(CamelCase, Thingy):
...     collection_name = "systemUsers"

>>> user = SystemUser.find_one()
>>> user.view()
{'_id': ObjectId(...), 'firstName': 'John', 'lastName': 'Doe'}

>>> user.first_name
'John'
>>> user.first_name = "Jonny"
>>> user.save()
SystemUser({'_id': ObjectId(...), firstName: 'Jonny', lastName: 'Doe'})
```

CHAPTER

THREE

TESTS

To run the tests suite:

- make sure you have a MongoDB database running on `localhost:27017` (you can spawn one with `docker-compose up -d`);
- install developers requirements with `pip install -r requirements.txt`;
- run `pytest`.

CHAPTER

FOUR

SPONSORS

CHAPTER

FIVE

DOCUMENTATION

5.1 API reference

class mongo_thingy.**AsyncThingy**(**args*, ***kwargs*)

async classmethod create_index(*keys*, ***kwargs*)

async classmethod create_indexes()

async classmethod find_one_and_replace(*filter*, *replacement*, **args*, ***kwargs*)

async classmethod find_one_and_update(*filter*, *update*, **args*, ***kwargs*)

async save(*force_insert=False*, *refresh=False*)

class mongo_thingy.**Thingy**(**args*, ***kwargs*)

classmethod create_index(*keys*, ***kwargs*)

classmethod create_indexes()

classmethod find_one_and_replace(*filter*, *replacement*, **args*, ***kwargs*)

classmethod find_one_and_update(*filter*, *update*, **args*, ***kwargs*)

save(*force_insert=False*, *refresh=False*)

mongo_thingy.**connect**(**args*, ***kwargs*)

mongo_thingy.**create_indexes**()

Create indexes registered on all *Thingy*

5.1.1 Cursor

class mongo_thingy.cursor.**AsyncCursor**(*delegate*, *thingy_cls=None*, *view=None*)

async delete()

async first()

next = <mongo_thingy.cursor._AsyncBindingProxy object>

to_list = <mongo_thingy.cursor._AsyncBindingProxy object>

class mongo_thingy.cursor.**Cursor**(*delegate*, *thingy_cls=None*, *view=None*)

delete()

first()

next = <mongo_thingy.cursor._BindingProxy object>

to_list(*length*)

5.1.2 Versioned

class mongo_thingy.versioned.**AsyncRevision**(**args*, ***kwargs*)

async save()

class mongo_thingy.versioned.**AsyncVersioned**

async delete(*author=None*)

async is_versioned()

async revert()

async save(*author=None*, ***kwargs*)

class mongo_thingy.versioned.**Revision**(**args*, ***kwargs*)

save()

class mongo_thingy.versioned.**Versioned**

delete(*author=None*)

is_versioned()

revert()

property revisions

save(*author=None*, ***kwargs*)

property version

property versioned

PYTHON MODULE INDEX

m

INDEX

T

V