yadm

Release 1.5.8

Dec 11, 2018

Contents

It's small and simple ODM for use with MongoDB.

CHAPTER 1

Requirements

YAMD support MongoDB version 3.x only. MongoDB 2.x is not supported.

Minimal version of python is 3.5.

CHAPTER 2

Quick start

2.1 Create the models

```
from datetime import datetime

import pymongo

from yadm import Database
from yadm import Document, EmbeddedDocument
from yadm import fields
from yadm.serialize import to_mongo


class User(Document):
    __collection__ = 'users'

    name = fields.StringField()
    email = fields.EmailField()


class PostLogItem(EmbeddedDocument):
    op = fields.StringField(choices=['created', 'comment_added'])
    at = fields.DatetimeField()
    data = fields.MongoMapField()


class Post(Document):
    __collection__ = 'posts'

    user = fields.ReferenceField(User)
    created_at = fields.DatetimeField(auto_now=True)
    title = fields.StringField()
    body = fields.StringField()
    log = fields.ListField(fields.EmbeddedDocumentField(PostLogItem))
```

(continues on next page)

(continued from previous page)

```
class Comment(Document):
    __collection__ = 'comments'

    user = fields.ReferenceField(User)
    created_at = fields.DatetimeField(auto_now=True)
    post = fields.ReferenceField(Post)
    text = fields.StringField()
```

All documents creates from class `Document`. You can use multiple inheritance.

`__collection__` magic attribute setups the collection name for documents of model.

2.2 Connect to database

```
client = pymongo.MongoClient('mongodb://localhost:27017')
db = Database(client, 'blog')
```

`Database` object is a wrapper about `pymongo` or `motor` `Database`.

2.3 Create documents

2.3.1 User

```
user = User(name='Bill', email='bill@galactic.hero')
db.insert(user)
```

Just insert document to database.

2.3.2 Post

```
post = Post()
post.user = user
post.title = 'Small post'
post.body = 'Bla-bla-bla...'
post.log = [PostLogItem(op='created', at=datetime.utcnow())]
db.insert(post)
```

You can fill documents as above.

2.3.3 Comment the post

```
comment = Comment()
comment.user = user
comment.post = post
comment.text = "RE: Bla-bla-bla..."
db.insert(comment)
```

(continues on next page)

(continued from previous page)

```
db.update_one(post, push={
    'log': to_mongo(PostLogItem(op='comment_added',
                                at=comment.created_at,
                                data={
                                  'comment': comment.id,
                                  'user': comment.user.id,
                                }))
})
```

We add log item to post's log. This is very usefull case.

2.4 Queries

2.4.1 find

```
qs = db(Post).find({'title': {'$regex': '^S'}})
assert qs.count() > 0
```

1. `db(Post)` creates the `QuerySet` object;
2. `find` method get the raw-query and return new `QuerySet` object with updated criteria;
3. `count` method make the query to database and return value.

```
for post in qs:
    assert post.title.startswith('S')
```

`__iter__` method make the `find`-query and returns the generator of documents.

2.4.2 find_one

```
post = db(Post).find_one({'user': user.id})
```

`find_one` get the first finded document.

```
user = post.user
```

Get attribute with reference makes the query to referred collection. Warning: N+1 problem! We have a cache in `QuerySet` object and get one referred document only once for one queryset.

2.4.3 join

Yes! We have a joins! In userspace...

```
comments = db(Comment).find({'post': post.id}).sort(('created_at', 1))
for comment in comments.join('user'):
    print(comment.user.name, comment.text)
```

1. Create the queryset for comments;
2. In `join` we get all documents for qs, get all users with one `$in`-query and bind to documents.

2.4.4 aggregations

```
agg = (db.aggregate(Comment)
       .match(user=user.id)
       .group(_id='post', count={'$sum': 1})
       .sort(count=-1))

for item in agg:
    print(item)
```

CHAPTER 3

CHANGES

3.1 1.5.0 (2017-12-31)

- Experimental `asyncio` support;
- **Add `ReferencesListField`** for lists of references.

3.2 1.4.15 (2017-12-27)

- **Add `projection` argument to `get_document`** and `reload`;
- Add `Document.__default_projection__` attribute.

3.3 1.4.14 (2017-11-06)

- **Add `EnumField`** for save `enum.Enum`;
- **Add `EnumStateField`** for simple state machines based on `enum.Enum`.

3.4 1.4.13 (2017-10-31)

- **Add `QuerySet.batch_size`** method for setup batch size for cursor;
- Some minor fixes.

3.5 1.4.10 (2017-07-07)

- **`ReferenceField.from_mongo`** try to get document from primary if not found by default.

3.6 1.4.9 (2017-07-06)

- **Add `QuerySet.read_primary` method** for simple setup `pymongo.read_preference.Primary`.

3.7 1.4.4 (2017-05-17)

- Add `TimedeltaField` for stores durations;
- **Add `SimpleEmbeddedDocumentField`** for simply create embedded documents.

```
class Doc(Document):
    embedded = SimpleEmbeddedDocumentField({
        'i': IntegerField(),
        's': StringField(),
    })
```

3.8 1.4.3 (2017-05-14)

- Add `StaticField` for static data.

3.9 1.4.2 (2017-04-09)

- Additional arguments (like `write_concern`) for write operations;
- `create_fake` save the documents with write concern "majority" by default.

3.10 1.4.0 (2017-04-05)

- Drop pymongo 2 support;
- Additional options for databases and collections;
- Add `Database.get_document`;
- Add `TypedEmbeddedDocumentField`;
- **`reload` argument of `Database.update_one` <yadm.database.Database.update_one>** must be keyword (may be backward incompotable).

3.11 1.3.1 (2017-02-21)

- Change raw data for `Money`;

3.12 1.3.0 (2017-02-19)

- **Add currency support to `Money`:**
  - Totaly rewrite `Money` type. Now it is not subclass of `Decimal`;
  - Add storage for currencies: `yadm.fields.money.currency.DEFAULT_CURRENCY_STORAGE`;

3.13 1.2.1 (2017-01-19)

- Add `QuerySet.find_in` for `$in` queries with specified order;

3.14 1.2.0 (2016-12-27)

- Drop MongoDB 2.X support;
- Objects for update and remove results;
- Use Faker instead fake-factory;

3.15 1.1.4 (2016-08-20)

- **Add some features to :py:module:‘Bulk <yadm.bulk>‘:**
  - `Bulk.update_one(document, **kw)`: method for add update one document in bulk;
  - `Bulk.find(query).update(**kw)`: update many documents by query;
  - `Bulk.find(query).upsert().update(**kw)`: upsert document;
  - `Bulk.find(query).remove(**kw)`: remove documents;

3.16 1.1.3 (2016-07-23)

- **Add `QuerySet.ids`** method for get only documents id’s from queryset;
- **Add `Money.total_cents`** method and `Money.total_cents` classmethod;

3.17 1.1 (2016-04-26)

- **Add cacheing on queryset level and use it for** `ReferenceField`;
- Add mongo aggregation framework support;
- **Add `exc` argument to** `QuerySet.find_one` for raise specified exception if not found;
- **Add `multi` argument to** `QuerySet.remove`;
- Deprecate `QuerySet.find_one`
- Refactoring.

3.18 1.0 (2015-11-14)

- **Change document structure. No more bad `BaseDocument.__data__` attribute:**
  - `BaseDocument.__raw__`: raw data from mongo;
  - `BaseDocument.__cache__`: cached objects, casted with fields;
  - `BaseDocument.__changed__`: changed objects.
- **Changes api for custom fields:**
  - Not more need create field descriptors for every field;
  - `prepare_value` called only for setattr;
  - `to_mongo` called only for save objects to mongo;
  - `from_mongo` called only for load values from `BaseDocument.__raw__`;
  - Remove *Field.default* attribute. Use `Field.get_default` method;
  - Add `get_if_not_loaded` and `get_if_attribute_not_set` method;
  - By default raise `NotLoadedError` if field not loaded from projection;
- **Changes in `ReferenceField`:**
  - Raise `BrokenReference` if link is bloken;
  - Raise `NotBindingToDatabase` if document not saved to database;
- *smart_null* keyword for `Field`;
- Fields in document must be instances (not classes!);
- Remove *ArrayContainer* and *ArrayContainerField*;
- Remove old *MapIntKeysField* and *MapObjectIdKeysField*. Use new `MapCustomKeysField`;
- Add `Database.update_one` method for run simple update query with specified document;
- Add `QuerySet.distinct`;
- `serialize.from_mongo` now accept *not_loaded* sequence with filed names who must mark as not loaded, *parent* and *name*;
- `serialize.to_mongo` do not call `FieldDescriptor.__set__`;
- Fakers! Subsystem for generate test objects;
- Tests now use pytest;
- And more, and more...

CHAPTER 4

API documentation

4.1 API

API documentation

4.1.1 Database

4.1.2 Documents

4.1.3 Serializers and deserializers

4.1.4 Queryset

4.1.5 Bulk queries

4.1.6 Mongo Aggregation Framework

4.1.7 Join

4.1.8 Fields

4.1.8.1 Base fields

4.1.8.2 Simple fields

4.1.8.3 Datetime field

4.1.8.4 Decimal field

4.1.8.5 Embedded documents fields

4.1.8.6 Reference field

4.1.8.7 Containers fields

4.1.8.8 List fields

4.1.8.9 Set field

4.1.8.10 Map field

4.1.8.11 Geo fields