Mnj

The MongoDB library which feels good

Serge Matveenko

assaia

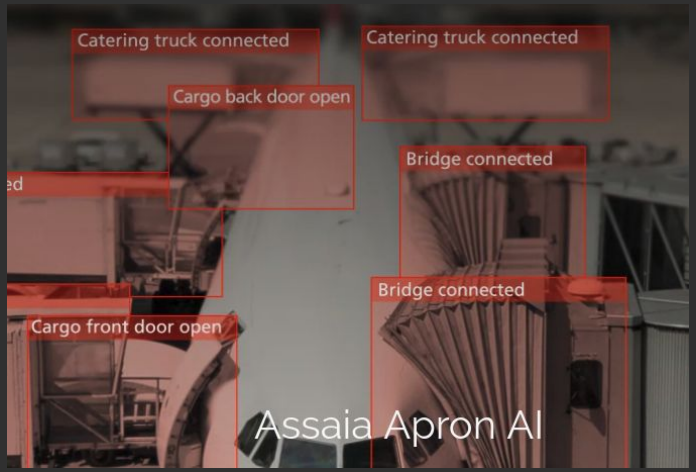
The Apron AI

About me

Serge Matveenko
github.com/lig
twitter.com/lig1

assaia.com

- 28 years with coding
- 12 years with Python
- 15+ programming languages
- FLOSS advocate
- MongoDB fan :)
- Software Architect @ Assaia



Why MongoDB?

Because “MongoDB Is Web Scale” (https://youtu.be/b2F-DItXtZs by gar1t)

MongoDB

mongodb.com

Features:

- Rich JSON Documents
- Powerful query language
- Aggregation Framework
- Full ACID transactions
- Support for joins in queries
- Replication & Automatic Failover
- Sharding & Location Segmentation
- File Storage aka GridFS

Dealing with MongoDB in Python

MongoDB & Python

- PyMongo — main driver
  - Direct MongoDB syntax mapping to dicts.
  - Officially supported
  - Has async derivative “Motor”
- MongoEngine — most popular ODM
  - Class to Collection mapping
  - Django-like syntax
  - Has async derivative “MotorEngine”
- Other
  - TxMongo (Twisted, PyMongo-like)
  - MongoKit (alternative ODM, lack of maintenance, slow, was buggy)
  - Some Django hacks, etc

MongoEngine

MongoEngine: almost as “good” as Django ORM

```
class User(Document):
    name = StringField()
class Page(Document):
    content = StringField()
    authors = ListField(ReferenceField(User))
...
Page(content="Test Page", authors=[bob, john]).save()
Page.objects(authors__in=[bob])
Page.objects(id='...').update_one(pull__authors=bob)
```

MongoEngine

Pros:

- Maps Classes to MognoDB Collections (ODM)
- Short learning curve: uses Django-like syntax

Cons:

- Django-like syntax lacks the power of MongoDB Query Language
- Django-like queries could become messy on complex sub-document queries
- All MongoEngine power is lost when you go around to pure PyMongo

He uses raw PyMongo with MongoEngine

PyMongo

PyMongo: everything is a dict

```
import pymongo


my_client = pymongo.MongoClient()
my_database = my_client['my_database']
my_collection = my_database['my_collection']


my_collection.find_one({'author': "William Gibson", 'title': "Count Zero"})
```

PyMongo: …even when it's not

```
import pymongo


my_client = pymongo.MongoClient()
my_database = my_client.my_database
my_collection = my_database.my_collection


my_collection.find_one({'author': "William Gibson", 'title': "Count Zero"})
```

PyMongo: …better with some tweaking

```
import pymongo


my_client = pymongo.MongoClient()
my_database = my_client['my_database']
my_collection: pymongo.collection.Collection = my_database['my_collection']


my_collection.find_one({'author': "William Gibson", 'title': "Count Zero"})
```

PyMongo: …results are still dicts

```
book = my_collection.find_one(
    {'author': "William Gibson", 'title': "Count Zero"}
)
book == {
    '_id': bson.ObjectId('5dae1a1d36fb82e1f046c8a3'),
    'author': "William Gibson",
    'title': "Count Zero",
    'year': 2006,
    'genre': ["Science Fiction", "Cyberpunk"],
}
```

PyMongo: …and it's getting worse

```
book = my_collection.find_one_and_update(
    {'_id': bson.ObjectId('5dae1a1d36fb82e1f046c8a3')},
    {'$push': {'genre': {'$each': ["Fiction"], '$position': 0}}},
    projection={'genre': 1},
    return_document=pymongo.ReturnDocument.AFTER,
)
book == {
    '_id': bson.ObjectId('5dae1a1d36fb82e1f046c8a3'),
    'genre': ["Fiction", "Science Fiction", "Cyberpunk"],
}
```

PyMongo: …and it's getting worse

```
book = my_collection.find_one_and_update(
    {'_id': bson.ObjectId('5dae1a1d36fb82e1f046c8a3')},
    {'$push': {'genre': {'$each': ["Fiction"], '$position': 0}}},
    projection={'genre': 1},
    return_document=pymongo.ReturnDocument.AFTER,
)
book == {
    '_id': bson.ObjectId('5dae1a1d36fb82e1f046c8a3'),
    'genre': ["Fiction", "Science Fiction", "Cyberpunk"],
}
```

That's how it makes me feel

Can we do something about it?

Mnj

github.com/lig/mnj

Goals:

- No strings attached (no pun intended)
- Classes are friends
- Dicts aren’t foes
- Hide routine things
- Do it the Python way
- Use the power of PyMongo
- Learn on SQLAlchemy, Peewee, Django ORM

Mnj: starting to feel better.

```
import nj


book = my_collection.find_one(
    nj.q(author="William Gibson", title="Count Zero")
)
book = my_collection.find_one_and_update(
    nj.q(_id=bson.ObjectId('5dae1a1d36fb82e1f046c8a3')),
    nj.push_(genre=nj.q(nj.each_(["Fiction"]), nj.position_(0))),
)
```

Mnj: starting to feel better..

```
import nj


book = my_collection.find_one(
    nj.q(author="William Gibson", title="Count Zero")
)
book = my_collection.find_one_and_update(
    nj.q(_id=bson.ObjectId('5dae1a1d36fb82e1f046c8a3')),
    nj.push_(genre=nj.q(nj.each_(["Fiction"]), nj.position_(0))),
)
```

Mnj: starting to feel better...

```
import nj


book = my_collection.find_one(
    nj.q(author="William Gibson", title="Count Zero")
)
book = my_collection.find_one_and_update(
    nj.q(_id=bson.ObjectId('5dae1a1d36fb82e1f046c8a3')),
    nj.push_(genre=(nj.each_(["Fiction"]), nj.position_(0))),
)
```

Mnj: now it feels good

```
class Book(nj.Document):
    author: str
    title: str
    year: int
    genre: typing.List[str]

book = Book.query(author="William Gibson", title="Count Zero").find_one()
book = Book.query(genre=nj.in_("Cyberpunk")).find_one()
```

Mnj: it feels good

```
Book.query(author="William Gibson", title="Count Zero").find_one()

Book.query.filter(title=nj.in_["Count Zero", "Neuromancer"]).find_one()

Book.query.find_one({'author': "William Gibson", 'title': "Count Zero"})

Book.query(author="William Gibson", title="Count Zero").update_one(
    nj.push_(genre=(nj.each_(["Fiction"]), nj.position_(0)))
)
```

He is excited already

Mnj: ODM how to

```
class MyDoc(dict):
    def __new__(cls, *args, **kwargs):
        print("__new__", args, kwargs)
        return super().__new__(cls, *args, **kwargs)
    def __setitem__(self, key, value):
        print("__setitem__", key, value)
        super().__setitem__(key, value)


MongoClient(document_class=MyDoc)
```

Mnj: ODM under the hood

```
class DocumentFactory(bson.raw_bson.RawBSONDocument, dict):
    def __new__(...):
        # Load the data from Raw BSON bytes
        # Figure out the namespace
        # Lookup the class we need in the registry
        # Build the instance
        return RegisteredDocumentClass(**data_loaded_from_bson)


MongoClient(document_class=DocumentFactory)
```

Mnj: the Client and the Client Manager

```
import nj


nj.create_client(db_name='books')
nj.create_client(db_name='archive', name='archive')


class OldBook(Book):
    class Meta:
        client_name = 'archive'
```

Mnj: you don't have to if you don't want to

```
my_client = nj.get_client(name='default')

nj.create_client(db_name='archive', name='archive', document_class=dict)
my_client = nj.get_client(name='archive')

my_client = pymongo.MongoClient(document_class=nj.DocumentFactory)

# Book._col: pymongo.collection.Collection
Book._col.find_one(...)
```

Mnj: Motor (asyncio) support

Should work with:

- Query builder
- Operator functions
- `Document._col` collection instance

Not yet in:

- Client Manager
- `Document.query` object

Mnj

github.com/lig/mnj

- Query Builder
- Operators as functions
- Object-Document Mapper
- Client/Connection Manager
- Document Factory for PyMongo

★ You don’t have to use everything

Mnj

github.com/lig/mnj

Coming soon:

- Smart operator helpers
- Even smarter ODM queries
- Server-side schema support
- Schema migrations: on-read, on-modification, immediate
- GridFS support
- Motor/asyncio support
- Flask plugin
- Other integrations
- …
- You use, you decide!

Thanks!

Questions?

Mnj:

- github.com/lig/mnj

Serge Matveenko:

- github.com/lig
- twitter.com/lig1
- keybase.io/lig