eBook Gratuit

APPRENEZ PyMongo

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#pymongo

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: pymongo

It is an unofficial and free PyMongo ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official PyMongo.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec PyMongo

Remarques

Cette section fournit une vue d'ensemble de ce qu'est pymongo et pourquoi un développeur peut vouloir l'utiliser.

Il devrait également mentionner tous les grands sujets au sein de pymongo, et établir un lien avec les sujets connexes. La documentation de pymongo étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Examples

Installation ou configuration

Instructions détaillées sur la mise en place ou l'installation de pymongo.

- Installation avec Pip
    - Pour installer pymongo pour la première fois:

      ```
      pip install pymongo
      ```

    - Installer une version spécifique de pymongo:

      Où XXX est la version à installer

      ```
      pip install pymongo==XXX
      ```

    - Mise à niveau du pymongo existant:

      ```
      pip install --upgrade pymongo
      ```

- Installation avec easy_install
    - Pour installer pymongo pour la première fois:

      ```
      python -m easy_install pymongo
      ```

    - Mise à niveau du pymongo existant:

      ```
      python -m easy_install -U pymongo
      ```

Bonjour le monde

PyMongo est un pilote Python natif pour MongoDB.

Installer PyMongo

```
pip install pymongo
```

Créer une connexion

Utilisez MongoClient pour créer une connexion. MongoClient utilise par défaut l'instance MongoDB s'exécutant sur `localhost:27017` si elle n'est pas spécifiée.

```
from pymongo import MongoClient
client = MongoClient()
```

Accéder aux objets de base de données

La classe Database de PyMongo représente la construction de base de données dans MongoDB. Les bases de données contiennent des groupes de collections liées de manière logique.

```
db = client.mydb
```

Accéder aux objets de collection

La classe Collection de PyMongo représente la construction de collection dans MongoDB. Les collections contiennent des groupes de documents connexes.

```
col = db.mycollection
```

MongoDB crée implicitement de nouvelles bases de données et collections lors de la première utilisation.

Fonctionnement de base du CRUD

MongoDB stocke les enregistrements de données sous forme de *documents* BSON . BSON est la représentation binaire de JSON.

```
$ python
>>> from pymongo import MongoClient
>>> client = MongoClient()
>>> col = client.mydb.test
```

Créer

Insérer un seul document `insert_one(document)`

```
>>> result = col.insert_one({'x':1})
>>> result.inserted_id
ObjectId('583c16b9dc32d44b6e93cd9b')
```

Insérer plusieurs documents `insert_many(documents)`

```
>>> result = col.insert_many([{'x': 2}, {'x': 3}])
>>> result.inserted_ids
[ObjectId('583c17e7dc32d44b6e93cd9c'), ObjectId('583c17e7dc32d44b6e93cd9d')]
```

Remplacez un seul document correspondant au filtre `replace_one(filter, replacement, upsert=False)` . (pour insérer un nouveau document si le document correspondant n'existe pas, utilisez `upsert=True`)

```
>>> result = col.replace_one({'x': 1}, {'y': 1})
>>> result.matched_count
1
>>> result.modified_count
1
```

Mettre à jour

Mettre à jour un seul document correspondant au filtre `update_one(filter, update, upsert=False)`

```
>>> result = col.update_one({'x': 1}, {'x': 3})
```

Mettre à jour un ou plusieurs documents correspondant au filtre `update_many(filter, update, upsert=False)`

```
>>> result = col.update_many({'x': 1}, {'x': 3})
```

Lis

Interroger la base de données `find(filter=None, projection=None, skip=0, limit=0, no_cursor_timeout=False)` . L'argument de *filtre* est un document prototype auquel tous les résultats doivent correspondre.

```
>>> result = col.find({'x': 1})
```

Obtenir un document unique à partir de la base de données `find_one(filter=None)`

```
>>> result = col.find_one()
```

Interroger avec projection

```
query={'x':1}
projection={'_id':0, 'x':1} # show x but not show _id
result=col.find(query,projection)
```

Effacer

Supprimer un seul document correspondant au filtre `delete_one(filter)`

```
>>> result = col.delete_one({'x': 1})
>>> result.deleted_count
1
```

Supprimer un ou plusieurs documents correspondant au filtre `delete_many(filter)`

```
>>> result = col.delete_many({'x': 1})
>>> result.deleted_count
3
```

PyMongo fournit également la fonctionnalité `find_one_and_delete()` , `find_one_and_update()` et `find_one_and_replace()` .

Lire Démarrer avec PyMongo en ligne: https://riptutorial.com/fr/pymongo/topic/2612/demarrer-avec-pymongo

Chapitre 2: Conversion entre BSON et JSON

Introduction

Dans de nombreuses applications, les enregistrements de MongoDB doivent être sérialisés au format JSON. Si vos enregistrements ont des champs de type date, datetime, objectId, binary, code, etc., vous rencontrez `TypeError: not JSON serializable` exceptions `TypeError: not JSON serializable` lors de l'utilisation de `json.dumps` . Cette rubrique montre comment surmonter cela.

Examples

Utiliser json_util

json_util fournit deux méthodes d'assistance, des `dumps` et des `loads` , qui encapsulent les méthodes json natives et fournissent une conversion BSON explicite vers et depuis json.

Usage simple

```
from bson.json_util import loads, dumps
record = db.movies.find_one()
json_str = dumps(record)
record2 = loads(json_str)
```

si `record` est:

```
{
    "_id" : ObjectId("5692a15524de1e0ce2dfcfa3"),
    "title" : "Toy Story 4",
    "released" : ISODate("2010-06-18T04:00:00Z")
}
```

alors `json_str` devient:

```
{
    "_id": {"$oid": "5692a15524de1e0ce2dfcfa3"},
    "title" : "Toy Story 4",
    "released": {"$date": 1276833600000}
}
```

JSONOptions

Il est possible de personnaliser le comportement des `dumps` au moyen d' un `JSONOptions` objet. Deux ensembles d'options sont déjà disponibles: `DEFAULT_JSON_OPTIONS` et `STRICT_JSON_OPTIONS` .

```
>>> bson.json_util.DEFAULT_JSON_OPTIONS
    JSONOptions(strict_number_long=False, datetime_representation=0,
```

```
    strict_uuid=False, document_class=dict, tz_aware=True,
    uuid_representation=PYTHON_LEGACY, unicode_decode_error_handler='strict',
    tzinfo=<bson.tz_util.FixedOffset object at 0x7fc168a773d0>)
```

Pour utiliser différentes options, vous pouvez:

1. modifiez l'objet `DEFAULT_JSON_OPTIONS` . Dans ce cas, les options seront utilisées pour tous les appels suivants aux `dumps` :

```
from bson.json_util import DEFAULT_JSON_OPTIONS
DEFAULT_JSON_OPTIONS.datetime_representation = 2
dumps(record)
```

2. spécifier un `JSONOptions` dans un appel à `dumps` utilisant le paramètre `json_options` :

```
# using strict
dumps(record, json_options=bson.json_util.STRICT_JSON_OPTIONS)

# using a custom set of options
from bson.json_util import JSONOptions
options = JSONOptions() # options is a copy of DEFAULT_JSON_OPTIONS
options.datetime_representation=2
dumps(record, json_options=options)
```

Les paramètres de `JSONOptions` sont les suivants:

- **strict_number_long** : Si true, les objets Int64 sont codés dans le type Strict NumberLong du mode Strict de MongoDB Extended JSON, c'est-à-dire `{"$numberLong": "<number>" }` . Sinon, ils seront encodés en int. La valeur par défaut est False.
- **datetime_representation** : représentation à utiliser lors du codage des instances de datetime.datetime. 0 => `{"$date": <dateAsMilliseconds>}` , 1 => `{"$date": {"$numberLong": "<dateAsMilliseconds>"}}` , 2 => `{"$date": "<ISO-8601>"}`
- **strict_uuid** : Si true, l'objet uuid.UUID est codé dans le type de mode Strict Binary de MongoDB Extended JSON. Sinon, il sera codé comme `{"$uuid": "<hex>" }` . La valeur par défaut est False.
- **document_class** : Les **documents** BSON renvoyés par load () seront décodés en une instance de cette classe. Doit être une sous-classe de collections.MutableMapping. Par défaut, dict.
- **uuid_representation** : représentation BSON à utiliser lors du codage et du décodage des instances de uuid.UUID. La valeur par défaut est PYTHON_LEGACY.
- **tz_aware** : Si la valeur est true, le type de mode Strict de MongoDB Extended JSON Date sera décodé en instances de datetime.datetime **identifiées** par le fuseau horaire. Sinon, ils seront naïfs. La valeur par défaut est True.
- **tzinfo** : Une sous-classe `datetime.tzinfo` qui spécifie le fuseau horaire à partir duquel les objets datetime doivent être décodés. La valeur par défaut est utc.

Utiliser python-bsonjs

python-bsonjs ne dépend pas de PyMongo et peut offrir une amélioration des performances par

rapport à `json_util` :

> bsonjs est environ 10 à 15 fois plus rapide que json_util de PyMongo pour décoder BSON en JSON et encoder JSON en BSON.

Notez que:

- pour utiliser efficacement bsonjs, il est recommandé de travailler directement avec `RawBSONDocument`
- les dates sont codées en utilisant la représentation LEGACY, c'est-à-dire `{"$date": <dateAsMilliseconds>}` . Il n'y a actuellement aucune option pour changer cela.

Installation

```
pip install python-bsonjs
```

Usage

Pour tirer pleinement parti des bsonjs, configurez la base de données pour utiliser la classe `RawBSONDocument` . Ensuite, utilisez `dumps` pour convertir des octets bruts bson en json et des `loads` pour convertir json en octets bruts bson:

```
import pymongo
import bsonjs
from pymongo import MongoClient
from bson.raw_bson import RawBSONDocument

# configure mongo to use the RawBSONDocument representation
db = pymongo.MongoClient(document_class=RawBSONDocument).samples
# convert json to a bson record
json_record = '{"_id": "some id", "title": "Awesome Movie"}'
raw_bson = bsonjs.loads(json_record)
bson_record = RawBSONDocument(raw_bson)
# insert the record
result = db.movies.insert_one(bson_record)
print(result.acknowledged)

# find some record
bson_record2 = db.movies.find_one()
# convert the record to json
json_record2 = bsonjs.dumps(bson_record2.raw)
print(json_record2)
```

Utilisation du module json avec des gestionnaires personnalisés

Si tout ce dont vous avez besoin est la sérialisation des résultats de mongo dans json, il est possible d'utiliser le module `json` , à condition que vous définissiez des gestionnaires personnalisés pour gérer les types de champs non sérialisables. Un avantage est que vous disposez de toute la puissance nécessaire pour encoder des champs spécifiques, tels que la représentation datetime.

Voici un gestionnaire qui code les dates en utilisant la représentation iso et l'ID sous forme de chaîne hexadécimale:

```
import pymongo
import json
import datetime
import bson.objectid

def my_handler(x):
    if isinstance(x, datetime.datetime):
        return x.isoformat()
    elif isinstance(x, bson.objectid.ObjectId):
        return str(x)
    else:
        raise TypeError(x)

db = pymongo.MongoClient().samples
record = db.movies.find_one()
# {u'_id': ObjectId('5692a15524de1e0ce2dfcfa3'), u'title': u'Toy Story 4',
#   u'released': datetime.datetime(2010, 6, 18, 4, 0),}

json_record = json.dumps(record, default=my_handler)
# '{"_id": "5692a15524de1e0ce2dfcfa3", "title": "Toy Story 4",
#    "released": "2010-06-18T04:00:00"}'
```

Lire Conversion entre BSON et JSON en ligne:
https://riptutorial.com/fr/pymongo/topic/9348/conversion-entre-bson-et-json

Chapitre 3: Filtrer les documents par heure de création stockée dans ObjectId

Introduction

Inclut des exemples de requêtes pymongo pour filtrer des documents par horodatage encapsulé dans ObjectId

Examples

Documents créés dans les 60 dernières secondes

Comment trouver des documents créés il y a 60 secondes

```
seconds = 60

gen_time = datetime.datetime.today() - datetime.timedelta(seconds=seconds)
dummy_id = ObjectId.from_datetime(gen_time)

db.CollectionName.find({"_id": {"$gte": dummy_id}})
```

Si vous vous trouvez dans un autre fuseau horaire, vous devrez peut-être compenser l'heure de la date et l'heure UTC

```
seconds = 60

gen_time = datetime.datetime.today() - datetime.timedelta(seconds=seconds)
# converts datetime to UTC
gen_time=datetime.datetime.utcfromtimestamp(gen_time.timestamp())

dummy_id = ObjectId.from_datetime(gen_time)

db.Collection.find({"_id": {"$gte": dummy_id}})
```

Lire Filtrer les documents par heure de création stockée dans ObjectId en ligne: https://riptutorial.com/fr/pymongo/topic/9855/filtrer-les-documents-par-heure-de-creation-stockee-dans-objectid

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec PyMongo	Community, Himavanth, Kheshav Sewnundun, tim
2	Conversion entre BSON et JSON	Derlin
3	Filtrer les documents par heure de création stockée dans ObjectId	Sawan Vaidya