EBook Gratis

APRENDIZAJE PyMongo

Free unaffiliated eBook created from
Stack Overflow contributors.

#pymongo

Tabla de contenido

Acerca de

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.

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

Capítulo 1: Empezando con PyMongo

Observaciones

Esta sección proporciona una descripción general de qué es pymongo y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de pymongo, y vincular a los temas relacionados. Dado que la Documentación para pymongo es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

Examples

Instalación o configuración

Instrucciones detalladas sobre cómo configurar o instalar pymongo.

- Instalar con pip
    - Para instalar pymongo por primera vez:

      ```
      pip install pymongo
      ```

    - Instalando una versión específica de pymongo:

      Donde XXX es la versión a instalar.

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

    - Actualizando pymongo existente:

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

- Instalar con easy_install
    - Para instalar pymongo por primera vez:

      ```
      python -m easy_install pymongo
      ```

    - Actualizando pymongo existente:

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

Hola Mundo

PyMongo es un controlador nativo de Python para MongoDB.

Instalar PyMongo

```
pip install pymongo
```

Crear una conexión

Utilice MongoClient para crear una conexión. MongoClient utiliza de forma predeterminada la instancia de MongoDB que se ejecuta en `localhost:27017` si no se especifica.

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

Acceder a objetos de base de datos

La clase Base de datos de PyMongo representa la construcción de la base de datos en MongoDB. Las bases de datos contienen grupos de colecciones lógicamente relacionadas.

```
db = client.mydb
```

Acceder a objetos de colección

La clase Colección de PyMongo representa la construcción de colección en MongoDB. Las colecciones contienen grupos de documentos relacionados.

```
col = db.mycollection
```

MongoDB crea nuevas bases de datos y colecciones implícitamente en el primer uso.

Funcionamiento básico de CRUD

MongoDB almacena los registros de datos como *documentos* BSON . BSON es la representación binaria de JSON.

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

Crear

Insertar un solo documento `insert_one(document)`

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

Insertar múltiples documentos `insert_many(documents)`

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

Reemplace un solo documento que coincida con el filtro `replace_one(filter, replacement, upsert=False)` . (para insertar un nuevo documento si no existe un documento coincidente, use `upsert=True`)

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

Actualizar

Actualice un solo documento que coincida con el filtro `update_one(filter, update, upsert=False)`

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

Actualice uno o más documentos que coincidan con el filtro `update_many(filter, update, upsert=False)`

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

Leer

Consulte la búsqueda de la base de datos `find(filter=None, projection=None, skip=0, limit=0, no_cursor_timeout=False)` . El argumento del *filtro* es un documento prototipo que todos los resultados deben coincidir.

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

Obtenga un solo documento de la base de datos `find_one(filter=None)`

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

Consulta Con Proyección

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

Borrar

Eliminar un solo documento que coincida con el filtro `delete_one(filter)`

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

Eliminar uno o más documentos que coincidan con el filtro `delete_many(filter)`

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

PyMongo también proporciona las `find_one_and_delete()` , `find_one_and_update()` y `find_one_and_replace()` .

Lea Empezando con PyMongo en línea:
https://riptutorial.com/es/pymongo/topic/2612/empezando-con-pymongo

Capítulo 2: Conversión entre BSON y JSON

Introducción

En muchas aplicaciones, los registros de MongoDB deben ser serializados en formato JSON. Si sus registros tienen campos de tipo fecha, fecha y hora, objectId, binario, código, etc., se encontrará con `TypeError: not JSON serializable` excepciones `TypeError: not JSON serializable` al usar `json.dumps` . Este tema muestra cómo superar esto.

Examples

Usando json_util

json_util proporciona dos métodos de ayuda, `dumps` y `loads` , que envuelven los métodos nativos de json y proporcionan una conversión BSON explícita desde y hacia json.

Uso simple

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

si el `record` es:

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

entonces `json_str` convierte en:

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

JSONOptions

Es posible personalizar el comportamiento de los `dumps` través de un objeto `JSONOptions` . Ya hay dos conjuntos de opciones disponibles: `DEFAULT_JSON_OPTIONS` y `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>)
```

Para usar diferentes opciones, puedes:

1. Modificar el objeto `DEFAULT_JSON_OPTIONS` . En este caso, las opciones se utilizarán para todas las llamadas posteriores a `dumps` :

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

2. especifique un `JSONOptions` en una llamada a `dumps` usando el parámetro `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)
```

Los parámetros de `JSONOptions` son:

- **strict_number_long** : Si es verdadero, los objetos Int64 están codificados en el tipo de modo Estricto JSON de MongoDB Extended JSON, es decir, `{"$numberLong": "<number>" }` . De lo contrario, serán codificados como un int. Por defecto es falso.
- **datetime_representation** : la representación a usar cuando se codifican instancias de datetime.datetime. 0 => `{"$date": <dateAsMilliseconds>}` , 1 => `{"$date": {"$numberLong": "<dateAsMilliseconds>"}}` , 2 => `{"$date": "<ISO-8601>"}`
- **strict_uuid** : Si es verdadero, el objeto uUidIDID se codifica en el modo Binario de modo estricto JSON de MongoDB. De lo contrario, se codificará como `{"$uuid": "<hex>" }` . Por defecto es falso.
- **document_class** : los **documentos** BSON devueltos por loads () se descodificarán a una instancia de esta clase. Debe ser una subclase de colecciones.MutableMapping. El valor predeterminado es dict.
- **uuid_representation** : la representación de BSON que se usará al codificar y decodificar instancias de uuidIDID. Por defecto es PYTHON_LEGACY.
- **tz_aware** : si es verdadero, la fecha del tipo de modo estricto de JSON extendido de MongoDB se decodificará a las instancias de datetime.datetime conscientes de la zona horaria. De lo contrario serán ingenuos. El valor predeterminado es True.
- **tzinfo** : una subclase de `datetime.tzinfo` que especifica la zona horaria desde la que se deben decodificar los objetos de fecha y hora. Por defecto es utc.

Usando python-bsonjs

python-bsonjs no depende de PyMongo y puede ofrecer una buena mejora de rendimiento sobre

`json_util` :

> bsonjs es aproximadamente 10-15 veces más rápido que json_util de PyMongo en la decodificación de BSON a JSON y la codificación de JSON a BSON.

Tenga en cuenta que:

- para usar bsonjs de manera efectiva, se recomienda trabajar directamente con `RawBSONDocument`
- las fechas se codifican utilizando la representación de LEGACY, es decir, `{"$date": <dateAsMilliseconds>}` . Actualmente no hay opciones para cambiar eso.

Instalación

```
pip install python-bsonjs
```

Uso

Para aprovechar al máximo los bsonjs, configure la base de datos para utilizar la clase `RawBSONDocument` . Luego, usa los `dumps` para convertir los bytes sin procesar de bson a json y las `loads` para convertir los bytes sin procesar de json a 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)
```

Usando el módulo json con manejadores personalizados.

Si todo lo que necesita es serializar los resultados de mongo en json, es posible usar el módulo `json` , siempre que defina controladores personalizados para tratar con los tipos de campos no serializables. Una ventaja es que tiene plena capacidad para codificar campos específicos, como la representación de fecha y hora.

Aquí hay un controlador que codifica fechas usando la representación iso y el id como una

cadena hexadecimal:

```
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"}'
```

Lea Conversión entre BSON y JSON en línea:
https://riptutorial.com/es/pymongo/topic/9348/conversion-entre-bson-y-json

Capítulo 3: Filtrar documentos por tiempo de creación almacenados en ObjectId

Introducción

Incluye ejemplos de consulta de pymongo para filtrar documentos por marca de tiempo encapsulada en ObjectId

Examples

Documentos creados en los últimos 60 segundos.

Cómo encontrar documentos creados hace 60 segundos.

```
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 se encuentra en una zona horaria diferente, es posible que deba desplazar datetime a 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}})
```

Lea Filtrar documentos por tiempo de creación almacenados en ObjectId en línea: https://riptutorial.com/es/pymongo/topic/9855/filtrar-documentos-por-tiempo-de-creacion-almacenados-en-objectid

Creditos

S. No	Capítulos	Contributors
1	Empezando con PyMongo	Community, Himavanth, Kheshav Sewnundun, tim
2	Conversión entre BSON y JSON	Derlin
3	Filtrar documentos por tiempo de creación almacenados en ObjectId	Sawan Vaidya