pymongo操作

要使用pymongo最先应该做的事就是先连上运行中的 mongod 。

导入 pymongo:

from pymongo import MongoClient

创建一个连接到 mongod 到客户端：

client = MongoClient()

或者：

client = MongoClient("mongodb://127.0.0.1:27019")

连接数据库:

假设要连接的数据库名为 primer

db = client.primer

或者：

db = client['primer']

连接到对应的数据集：

coll = db.dataset

coll = db['dataset']

至此，已经完整对连接了数据库和数据集，完成了初识化的操作。

2. 插入数据

insert_one(document)

insert_many(documents, ordered=True)

- insert_one(document)

```
from datetime import datetime
result = db.restaurants.insert_one(
  {
    "address": {
      "street": "2 Avenue",
      "zipcode": "10075",
      "building": "1480",
      "coord": [-73.9557413, 40.7720266]
    },
```

```
    "borough": "Manhattan",
    "cuisine": "Italian",
    "grades": [
      {
        "date": datetime.strptime("2014-10-01", "%Y-%m-%d"),
        "grade": "A",
        "score": 11
      },
      {
        "date": datetime.strptime("2014-01-16", "%Y-%m-%d"),
        "grade": "B",
        "score": 17
      }
    ],
    "name": "Vella",
    "restaurant_id": "41704620"
  }
)
```

其中inserted_id### 是插入的元素多_id### 值。

- insert_many(documents, ordered=True)

```
result = db.test.insert_many([{'x': i} for i in range(2)])
```

查询数据

find(filter=None, projection=None, skip=0, limit=0, no_cursor_timeout=False, cursor_type=CursorType.NON_TAILABLE, sort=None, allow_partial_results=False, oplog_replay=False, modifiers=None, manipulate=True)
find_one(filter_or_id=None, *args, **kwargs)

- find

- ### find 查询出来的是一个列表集合。

```
cursor = db.restaurants.find()
for document in cursor:
  print(document)
```

查询字段是最上层的

cursor = db.restaurants.find({“borough”: “Manhattan”})

查询字段在内层嵌套中

cursor = db.restaurants.find({“address.zipcode”: “10075”})

- ### 操作符查询

cursor = db.restaurants.find({“grades.score”: {“$gt”: 30}})

cursor = db.restaurants.find({“grades.score”: {“$lt”: 10}})

AND

cursor = db.restaurants.find({“cuisine”: “Italian”, “address.zipcode”: “10075”})

cursor = db.restaurants.find(

{“$or”: [{“cuisine”: “Italian”}, {“address.zipcode”: “10075”}]})

- find_one

返回的是一个JSON式文档，所以可以直接使用！

- sort

- ### 排序时要特别注意，使用的并不是和mongo shell的一样，而是使用了列表，
- ### 当排序的标准只有一个，且是递增时，可以直接写在函数参数中：

pymongo.ASCENDING = 1

pymongo.DESCENDING = -1

cursor = db.restaurants.find().sort(“borough”)

cursor = db.restaurants.find().sort([

(“borough”, pymongo.ASCENDING),

(“address.zipcode”, pymongo.DESCENDING)

])

更新文档

更新文档的函数有三个(不能更新_id字段)

update_one(filter, update, upsert=False)

update_many(filter, update, upsert=False)

replace_one(filter, replacement, upsert=False)

find_one_and_update(filter, update, projection=None, sort=None,

return_document=ReturnDocument.BEFORE, **kwargs)

update_one

返回结果是一个：UpdateResult ，如果查找到多个匹配，则只更新第一个！

```
result = db.restaurants.update_one(
  {“name”: “Juni”},
  {
    “$set”: {
      “cuisine”: “American (New)”
    },
    “$currentDate”: {“lastModified”: True}
  }
)
```

update_many

查找到多少匹配，就更新多少。

```
result = db.restaurants.update_many(
  {“address.zipcode”: “10016”, “cuisine”: “Other”},
  {
    “$set”: {“cuisine”: “Category To Be Determined”},
    “$currentDate”: {“lastModified”: True}
  }
)
```

删除文档

删除一个

```
result = db.test.delete_one({‘x’: 1})
```

删除多个

```
result = db.restaurants.delete_many({“borough”: “Manhattan”})
```

删除全部

```
result = db.restaurants.delete_many({})
```

删除整个集合，是drop_collection()的别名

```
db.restaurants.drop()
```