**Python连接MongoDB操作**

**1.安装PyMongo**

注意：请勿安装“bson”软件包。 PyMongo配有自己的bson包; 执行“pip install bson”或“easy\_install bson”则会安装与PyMongo不兼容的第三方软件包。

**使用pip安装**

我们建议在所有平台上使用pip来安装pymongo：

C:\Users\Administrator>python -m pip install pymongo

Collecting pymongo

Downloading pymongo-3.4.0.tar.gz (583kB)

... ....

Installing collected packages: pymongo

Running setup.py install for pymongo ... done

Successfully installed pymongo-3.4.0

Shell

要获得pymongo的特定版本：

$ python -m pip install pymongo==3.1.1

Shell

要升级pymongo的版本：

$ python -m pip install --upgrade pymongo

Shell

**Python版本依赖**

PyMongo支持CPython 2.6,2.7,3.3+，PyPy和PyPy3。

GSSAPI和TLS的可选依赖关系：

GSSAPI认证需要Windows上的Unix或WinKerberos上的pykerberos。PyMongo可以自动安装正确的依赖关系：

$ python -m pip install pymongo[gssapi]

Shell

**2.使用MongoClient建立连接**

使用PyMongo时，第一步是运行 mongod 实例创建一个MongoClient。如下：

from pymongo import MongoClient

client = MongoClient()

Python

上述代码将连接默认主机和端口。 也可以明确指定主机和端口，如下所示：

from pymongo import MongoClient

#client = MongoClient()

client = MongoClient('localhost', 27017)

或使用MongoDB URI格式：

client = MongoClient('mongodb://localhost:27017/')

**3.获取数据库**

MongoDB的一个实例可以支持多个独立的数据库。 在使用PyMongo时，可以使用MongoClient实例上的属性的方式来访问数据库：

db = client.pythondb

如果数据库名称使用属性方式访问无法正常工作(如：python-db)，则可以使用字典样式访问：

db = client['python-db']

**4.获取集合**

集合是存储在MongoDB中的一组文档，可以类似于关系数据库中的表。 在PyMongo中获取集合的工作方式与获取数据库相同：

collection = db.python\_collection

Shell

或(使用字典方式访问)：

collection = db['python-collection']

Shell

MongoDB中关于集合(和数据库)的一个重要注意事项是它们是懒创建的 - 上述任何命令都没有在MongoDB服务器上实际执行任何操作。当第一个文档插入集合时才创建集合和数据库。

集合是存储在MongoDB中的一组文档，可以被认为大致相当于关系数据库中的表。 在PyMongo中获取集合的工作方式与获取数据库相同：

**5.文档**

MongoDB中的数据使用JSON方式来表示文档(并存储)。 在PyMongo中使用字典来表示文档。例如，以下字典可能用于表示博客文章：

import datetime

from pymongo import MongoClient

client = MongoClient()

post = {"author": "Mike",

"text": "My first blog post!",

"tags": ["mongodb", "python", "pymongo"],

"date": datetime.datetime.utcnow()}

`

Python

**6.插入文档**

要将文档插入到集合中，可以使用insert\_one()方法：

#!/usr/bin/python3

#coding=utf-8

import datetime

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

post = {"author": "Maxsu",

"text": "My first blog post!",

"tags": ["mongodb", "python", "pymongo"],

"date": datetime.datetime.utcnow()}

posts = db.posts

post\_id = posts.insert\_one(post).inserted\_id

print ("post id is ", post\_id)

Python

执行上面代码，得到以下结果 -

post id is 595965fe4959eb09c4451091

Shell

插入文档时，如果文档尚未包含“\_id”键，则会自动添加“\_id”。 “\_id”的值在集合中必须是唯一的。 insert\_one()返回一个InsertOneResult的实例。 有关“\_id”的更多信息，请参阅[有关\_id文档](http://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.insert_one)。

插入第一个文档后，实际上已经在服务器上创建了帖子(posts)集合。可以列出数据库中的所有集合：

#!/usr/bin/python3

#coding=utf-8

import datetime

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

"""

post = {"author": "Maxsu",

"text": "My first blog post!",

"tags": ["mongodb", "python", "pymongo"],

"date": datetime.datetime.utcnow()}

posts = db.posts

post\_id = posts.insert\_one(post).inserted\_id

print ("post id is ", post\_id)

"""

#cur\_collection = db.collection\_names(include\_system\_collections=False) #这个方法过时了，使用下面的方法

cur\_collection = db.list\_collection\_names(include\_system\_collections=False)

print("cur\_collection is :", cur\_collection)

Python

执行上面代码，得到以下结果 -

cur\_collection is : ['posts']

Shell

**7.使用find\_one()获取单个文档**

MongoDB中执行的最基本的查询类型是find\_one()。 此方法返回与查询匹配的单个文档(如果没有匹配，则返回None)。 当知道只有一个匹配的文档，或只对第一个匹配感兴趣时则可考虑使用find\_one()方法。下面示例中使用find\_one()从帖子(posts)集中获取第一个文档：

#!/usr/bin/python3

#coding=utf-8

import datetime

import pprint

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

'''

post = {"author": "Maxsu",

"text": "My first blog post!",

"tags": ["mongodb", "python", "pymongo"],

"date": datetime.datetime.utcnow()}

'''

posts = db.posts

#post\_id = posts.insert\_one(post).inserted\_id

#print ("post id is ", post\_id)

pprint.pprint(posts.find\_one())

Python

执行上面代码，得到以下结果 -

{'\_id': ObjectId('595965fe4959eb09c4451091'),

'author': 'Maxsu',

'date': datetime.datetime(2017, 7, 2, 21, 30, 38, 402000),

'tags': ['mongodb', 'python', 'pymongo'],

'text': 'My first blog post!'}

Shell

结果是匹配之前插入的字典格式(Json)。注意: 返回的文档包含一个“\_id”，它是在插入时自动添加的。

find\_one()方法还支持查询结果文档必须匹配的特定元素。要查询作者是“Maxsu”的文档，可以指定查询的条件，如下所示：

#!/usr/bin/python3

#coding=utf-8

import datetime

import pprint

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

post = {"author": "Minsu",

"text": "This blog post belong to Minsu!",

"tags": ["MySQL", "Oracle", "pymongo"],

"date": datetime.datetime.utcnow()}

posts = db.posts

post\_id = posts.insert\_one(post).inserted\_id

post = posts.find\_one({"author": "Maxsu"})

pprint.pprint(post)

#print (post)

Python

执行上面代码，得到以下结果 -

{'\_id': ObjectId('595965fe4959eb09c4451091'),

'author': 'Maxsu',

'date': datetime.datetime(2017, 7, 2, 21, 30, 38, 402000),

'tags': ['mongodb', 'python', 'pymongo'],

'text': 'My first blog post!'}

Shell

**8.通过ObjectId查询**

也可以通过它的\_id找到一个帖子(post)，下面的示例子中演示如何根据给定的一个ObjectId查询数据：

#!/usr/bin/python3

#coding=utf-8

import datetime

import pprint

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

post = {"\_id": 100,

"author": "Kuber",

"text": "This is is my first post!",

"tags": ["Docker", "Shell", "pymongo"],

"date": datetime.datetime.utcnow()}

posts = db.posts

post\_id = posts.insert\_one(post).inserted\_id

print("post\_id is :", post\_id)

post = posts.find\_one({"\_id": post\_id})

print("Find By Post ID:")

pprint.pprint(post)

#print (post)

Python

执行上面代码，得到以下结果 -

post\_id is : 100

Find By Post ID:

{'\_id': 100,

'author': 'Kuber',

'date': datetime.datetime(2017, 7, 3, 14, 14, 8, 28000),

'tags': ['Docker', 'Shell', 'pymongo'],

'text': 'This is is my first post!'}

Shell

Web应用程序中的常见任务是从请求URL获取ObjectId并找到匹配的文档。 在这种情况下，必须将ObjectId从一个字符串转换到find\_one()：

from bson.objectid import ObjectId

# The web framework gets post\_id from the URL and passes it as a string

def get(post\_id):

# Convert from string to ObjectId:

document = client.db.collection.find\_one({'\_id': ObjectId(post\_id)})

Python

**9.关于Unicode字符串的注释**

您可能已经注意到，我们先前存储的常规Python字符串在从服务器检索时看起来是不同的(例如，u’Mike而不是“Mike”)。一个简短的解释是有序的字符串。

MongoDB以BSON格式存储数据。BSON字符串是UTF-8编码的，所以PyMongo必须确保它存储的任何字符串只包含有效的UTF-8数据。 常规字符串(<type'str'>)被验证并保存不变。 Unicode字符串(<type'unicode'>)首先被编码为UTF-8。 我们的示例字符串在Python shell中表示为u'Mike而不是“Mike”的原因是PyMongo将每个BSON字符串解码为Python unicode字符串，而不是常规str。

**10.批量插入**

为了执行更复杂一些的查询，我们再插入一些文档。 除了插入单个文档外，还可以通过将列表作为第一个参数传递给insert\_many()来执行批量插入操作。 这将在列表中插入每个文档，只向服务器发送一个命令：

#!/usr/bin/python3

#coding=utf-8

import datetime

import pprint

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

new\_posts = [{"\_id": 1000,

"author": "Curry",

"text": "Another post!",

"tags": ["bulk", "insert"],

"date": datetime.datetime(2017, 11, 12, 11, 14)},

{"\_id": 1001,"author": "Maxsu",

"title": "MongoDB is fun",

"text": "and pretty easy too!",

"date": datetime.datetime(2019, 11, 10, 10, 45)}]

posts = db.posts

result = posts.insert\_many(new\_posts)

print("Bulk Inserts Result is :", result.inserted\_ids)

#print (post)

Python

执行上面代码，得到以下结果 -

Bulk Inserts Result is : [1000, 1001]

Shell

有几个有趣的事情要注意这个例子：

* insert\_many()的结果现在返回两个ObjectId实例，每个ID表示插入的一个文档。
* new\_posts[1]具有与其他帖子不同的“形状”(数据结构) - 没有“tags”字段，添加了一个新字段“title”。MongoDB是无模式的，表示的就是这个意思。

**11.查询多个文档**

要查询获得超过单个文档作为查询的结果，可使用find()方法。find()返回一个Cursor实例，它允许遍历所有匹配的文档。如下示例，遍历帖子集合中的每个文档：

#!/usr/bin/python3

#coding=utf-8

import datetime

import pprint

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

posts = db.posts

for post in posts.find():

pprint.pprint(post)

Python

执行上面代码，得到以下结果 -

{'\_id': ObjectId('595965fe4959eb09c4451091'),

'author': 'Maxsu',

'date': datetime.datetime(2017, 7, 2, 21, 30, 38, 402000),

'tags': ['mongodb', 'python', 'pymongo'],

'text': 'My first blog post!'}

{'\_id': 100,

'author': 'Kuber',

'date': datetime.datetime(2017, 7, 3, 14, 14, 8, 28000),

'tags': ['Docker', 'Shell', 'pymongo'],

'text': 'This is is my first post!'}

{'\_id': 1000,

'author': 'Curry',

'date': datetime.datetime(2017, 11, 12, 11, 14),

'tags': ['bulk', 'insert'],

'text': 'Another post!'}

{'\_id': 1001,

'author': 'Maxsu',

'date': datetime.datetime(2019, 11, 10, 10, 45),

'text': 'and pretty easy too!',

'title': 'MongoDB is fun'}

Shell

类似使用find\_one()一样，我们可以将文档传递给find()来限制返回的结果。 在这里，只希望得到作者是“Maxsu”的文档：

#!/usr/bin/python3

#coding=utf-8

import datetime

import pprint

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

posts = db.posts

for post in posts.find({"author": "Maxsu"}):

pprint.pprint(post)

Python

执行上面的代码，得到以下结果 -

{'\_id': ObjectId('595965fe4959eb09c4451091'),

'author': 'Maxsu',

'date': datetime.datetime(2017, 7, 2, 21, 30, 38, 402000),

'tags': ['mongodb', 'python', 'pymongo'],

'text': 'My first blog post!'}

{'\_id': 1001,

'author': 'Maxsu',

'date': datetime.datetime(2019, 11, 10, 10, 45),

'text': 'and pretty easy too!',

'title': 'MongoDB is fun'}

Shell

**12.计数统计**

如果只想知道有多少文档匹配查询，可以执行count()方法操作，而不是一个完整的查询。 可以得到一个集合中的所有文档的计数：

#!/usr/bin/python3

#coding=utf-8

import datetime

import pprint

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

posts = db.posts

print("posts count is = ", posts.count())

print("posts's author is Maxsu count is =", posts.find({"author": "Maxsu"}).count())

Python

执行上面代码，得到以下结果 -

posts count is = 4

posts's author is Maxsu count is = 2

Shell

**13.范围查询**

MongoDB支持许多不同类型的高级查询。例如，可以执行一个查询，将结果限制在比特定日期更早的帖子，而且还可以按作者对结果进行排序：

#!/usr/bin/python3

#coding=utf-8

import datetime

import pprint

from pymongo import MongoClient

client = MongoClient()

db = client.pythondb

posts = db.posts

d = datetime.datetime(2019, 11, 12, 12)

for post in posts.find({"date": {"$lt": d}}).sort("author"):

pprint.pprint(post)

Python

这里使用特殊的“$lt”运算符做范围查询，并且还可以调用sort()来按作者对结果进行排序。

**14.索引**

添加索引可以帮助加速某些查询，并且还可以添加额外的功能来查询和存储文档。在这个例子中，将演示如何在一个键上创建一个唯一的索引，该索引将拒绝已经存在值的文档插入。

首先，我们创建索引：

result = db.profiles.create\_index([('user\_id', pymongo.ASCENDING)], unique=True)

sorted(list(db.profiles.index\_information()))

Python

请注意，现在有两个索引：一个是MongoDB自动创建的在\_id索引，另一个是刚刚创建在user\_id上的索引。

现在来设置一些用户配置文件：

user\_profiles = [{'user\_id': 211, 'name': 'Luke'},{'user\_id': 212, 'name': 'Ziltoid'}]

result = db.profiles.insert\_many(user\_profiles)

Python

该索引将阻止 user\_id 已经在集合中的文档插入：

new\_profile = {'user\_id': 213, 'name': 'Drew'}

duplicate\_profile = {'user\_id': 212, 'name': 'Tommy'}

result = db.profiles.insert\_one(new\_profile) # This is fine.

result = db.profiles.insert\_one(duplicate\_profile)

## 出现错误提示...

Traceback (most recent call last):

DuplicateKeyError: E11000 duplicate key error index: test\_database.profiles.$user\_id\_1 dup key: { : 212 }

Python