**PyMongo学习**

# 安装

pip install pymongo

# 使用

pymongo 是python官方发行的mongodb库

官网地址：<http://api.mongodb.com/python/current/>

自带连接池，默认大概1000连接，每个连接缓存是10m

当前版本3.6.1

## 2.1 连接

(1) 导入

from pymongo import MongoClient

(2) 连接mongodb

host = localhost

port = 27017

client = MongoClient(host,port)

(3) 连接数据库

db = client.test\_database

or

db = client[test\_database]

注：当连接的数据库设置了auth之后，需要这样连接，不同版本，不同平台都不一样，具体见官方文档

client[test\_database].authenticate("user","passwd",mechanism="SCRAM-SHA-1")

(4) 连接数据库中的某个集合

## 获取该数据库中的非系统集合

db.collection\_names(include\_system\_collections=False)

## 连接集合

collection = db[collection\_name]

or

collection = db.collection\_name

## 2.2 增删改查

很好的一点就是，数据库不需要先建立，在连接后，如果进行插入数据操作，系统可以自己创建，上面提到过，mongodb的数据，使用的是类似json风格的文档，在python中即为字典，形如下例：

insert\_data = {

"author": "Mike",

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

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

"date": datetime.datetime.utcnow()

}

collection = db[collection\_name]

### 增

在插入一个文档时，MongoDB会自动给每个文档增加一个”\_id”的键，这个键是通过复杂计算出来的，不会重复，类似于下面这样的：

ObjectId('4ea02dfdd483050fe8000001')

那么如何利用ObjectId查询，这里有一个坑

>>> import pymongo

>>> import time

>>> db = pymongo.Connection("192.168.xx.xx",27017).linuxyan

>>> posts = db.posts

>>> post = {"id": "1",

... "author": "Mike",

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

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

... "date": time.strftime('%Y-%m-%d %H:%M:%S')}

>>> posts.insert(post)

ObjectId('53bd5a5fe138235f74b67563')

#//插入数据成功

#//利用{'author':'Mike'} 测试查询正常

>>> posts.find\_one({'author':'Mike'})

{u'\_id': ObjectId('53bd5a5fe138235f74b67563'), u'author': u'Mike',....}

#//利用{'\_id':"ObjectId('53bd5a5fe138235f74b67563')"}查询为空

>>>posts.find\_one({'\_id':"ObjectId('53bd5a5fe138235f74b67563')"})

#//如何利用ObjectId来查询？

>>> from bson import ObjectId

>>> posts.find\_one({'\_id':ObjectId('53bd5a5fe138235f74b67563')})

{ u'\_id': ObjectId('53bd5a5fe138235f74b67563'), u'author': u'Mike',....}

#//原来ObjectId是一个对象，而不是一个字符串

增加数据不需要事先定义文档的机构，每个文档的结构也可以不一样

1. 单条插入

result = collection.insert\_one(insert\_data)

1. 多条插入

result = collection.insert\_many(insert\_datas)

1. 返回插入id

result.insert\_id [单]

result.insert\_ids [多]

### (2) 查

1. 无条件查询

collection.find\_one() [单]

collection.find() [多]

1. 条件查询

collection.find({"author": "Mike"})

collection.find\_one({"author": "Mike"})

cursor = collection.find({

"$or":[{"cuisine": "Italian", "address.zipcode": "10075"}]

}

)

[仅显示cuisine为Italian和address.zipcode为10075的数据。两个条件只要满足一个就可以]

1. 区间查询

d = datetime.datetime(2009, 11, 12, 12)

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

print post

$lt 小于

1. 文档记录数

collection.count()

1. 数据排序

调用 pymongo.ASCENDING()和pymongo.DESCENDING()来指定是按升降序进行排序

cursor = collection.find().sort([

("borough", pymongo.ASCENDING),

("address.zipcode", pymongo.ASCENDING)

])

1. 从第几行开始读取(SLICE)，读取多少行(LIMIT)

for u in collection.find().skip(2).limit(3):

print u

1. IN

for u in db.users.find({"age":{"$in":(23, 26, 32)}}): print u

for u in db.users.find({"age":{"$nin":(23, 26, 32)}}): print u

1. 是否存在

db.users.find({'sex':{'$exists':True}}) # select \* from 集合名 where exists 键1

db.users.find({'sex':{'$exists':False}}) # select \* from 集合名 where not exists 键1

* 1
* 2

1. 正则表达式查询

for u in db.users.find({"name" : {"$regex" : r"(?i)user[135]"}}, ["name"]): print u # 查询出 name 为 user1, user3, user5 的

1. 多级路径的元素值匹配

Document 采取 JSON-like 这种层级结构，因此我们可以直接用嵌入(Embed)代替传统关系型数据库的关联引用(Reference)。

MongoDB 支持以 "." 分割的 namespace 路径，条件表达式中的多级路径须用引号

# 条件表达式中的多级路径须用引号,以 "." 分割

u = 集合名.find\_one({"im.qq":12345678})

# 查询结果如：{"\_id" : ObjectId("4c479885089df9b53474170a"), "name" : "user1", "im" : {"msn" : "user1@hotmail.com", "qq" : 12345678}}

### (3) 改

origin\_data = {"title":"Python and MongoDB",

"slug":"python-mongodb",

"author":"SErHo",

"content":"Python and MongoDB....",

"tags":["Python","MongoDB"],

"time":datetime.datetime.now()}

>>> post = posts.find\_one({"slug":"python-mongodb"})

>>> post["author"]

u'SErHo'

>>> post["author"] = "HaHa Lu"

>>> post["title"] = "Test Update"

>>> post["title"] = "Test Update"

>>> post["\_id"]

ObjectId('4ea0207dd483050fe8000001')

>>> posts.update({"\_id":post["\_id"]},post)

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

>>> print post

{u'author': u'HaHa Lu', u'title': u'Test Update',

u'tags': [u'Python', u'MongoDB'],

u'content': u'Python and MongoDB....',

u'time': datetime.datetime(2011, 10, 20, 21, 21, 52, 818000),

u'\_id': ObjectId('4ea0207dd483050fe8000001'),

u'slug': u'python-mongodb'}

首先我们根据slug来获得一篇文章，然后可以通过Python字典访问方法得到键的值，然后重新设置，再对post集合进行更新，在对整个集合进行更新时，你得先匹配要更改的文档，利用\_id这个属性来更新是比较常用的方法，因为你其他改了，这个可改不了。在执行update中最常见的错误就是限制的条件找到了多个文档，如果这样，数据库就不会更新这个集合，所有最好使用\_id来匹配。

如果只更新一个键呢，那就不用这么大费周折了，可以使用”$set”这个修改器，指定一个键，如果不存在，就可以创建。比如我要继续更新上面那篇文章的content，可以这样做（记住，修改它，必须先找到它，这里我利用上面查询到的\_id值来找）：

>>> posts.update({"\_id":post["\_id"]},{"$set": {"content":"Test Update SET...."}})

MongoDB的修改是很强大的，你可以把数据类型也给改了，比如把tags的数组改成普通的字符串。”set”过后又想删除这个键，可以使用”set”过后又想删除这个键，可以使用”unset”。如果我的这个post里面有一个键是views，即文章访问的次数，我想在每次访问这个文章后给它的值增加1，这该怎么办？于是”$inc”修改器出场了，这个可以用来增加已有键的值，如果没有，则创建它，类似的用法是：

>>> posts.update({"\_id":post["\_id"]},{"$inc": {"views":1}})

如果想修改tags这个数组里面的内容怎么办？有一个办法就是用set整体修改，但只是改里面的一些元素呢，MongoDB准备好了用于数组的修改器。比如，想要在tags里面加一个”Test”，这需要使用”set整体修改，但只是改里面的一些元素呢，MongoDB准备好了用于数组的修改器。比如，想要在tags里面加一个”Test”，这需要使用”push”，它可以在数组末尾添加一个元素：

>>> posts.update({"\_id":post["\_id"]},{"$push":{"tags":"Test"}})

为了避免加入了重复的，可以将”push”改为使用”push”改为使用”addToSet”，如果需要添加多个值，可以配合”$each”来使用，这样就可以添加不重复的进去，如下面：

>>> posts.update({"\_id":post["\_id"]},{"$addToSet": {"tags":{"$each":["Python","Each"]}}})

### (4) 删

>>> posts.delete\_one({"\_id":post["\_id"]})

# 相关操作

注：collection:user

(1) $all: 判断数组属性是否包含全部条件。

db.users.insert({'name':"user3", 'data':[1,2,3,4,5,6,7]})

db.users.insert({'name':"user4", 'data':[1,2,3]})

for u in db.users.find({'data':{'$all':[2,3,4]}}): print u

# 显示： { "\_id" : ObjectId("4c47a133b48cde79c6780df0"), "name" : "user3", "data" : [ 1, 2, 3, 4, 5, 6, 7 ] }

注意和 $in 的区别。$in 是检查目标属性值是条件表达式中的一员，而 $all 则要求属性值包含全部条件元素。

(2) $size: 匹配数组属性元素数量。

for u in db.users.find({'data':{'$size':3}}): print u

# 只显示匹配此数组数量的： { "\_id" : ObjectId("4c47a13bb48cde79c6780df1"), "name" : "user4", "data" : [ 1, 2, 3 ] }

(3) $type: 判断属性类型。

for u in db.users.find({'t':{'$type':1}}): print u # 查询数字类型的

for u in db.users.find({'t':{'$type':2}}): print u # 查询字符串类型的

类型值:

double:1

string: 2

object: 3

array: 4

binary data: 5

object id: 7

boolean: 8

date: 9

null: 10

regular expression: 11

javascript code: 13

symbol: 14

javascript code with scope: 15

32-bit integer: 16

timestamp: 17

64-bit integer: 18

min key: 255

max key: 127

(4) $not: 取反，表示返回条件不成立的文档。

似乎只能跟正则和 $mod 一起使用？？？？

# 还不知如何使用

(5) $unset: 和 $set 相反，表示移除文档属性。

for u in db.users.find({'name':"user1"}): print u

# 显示如： { "\_id" : ObjectId("4c479885089df9b53474170a"), "name" : "user1", "age" : 15, "address" : [ "address1", "address2" ] }

db.users.update({'name':"user1"}, {'$unset':{'address':1, 'age':1}})

for u in db.users.find({'name':"user1"}): print u

# 显示如： { "\_id" : ObjectId("4c479885089df9b53474170a"), "name" : "user1" }

(6) $push: 和 $ pushAll 都是向数组属性添加元素。# 好像两者没啥区别

for u in db.users.find({'name':"user1"}): print u

# 显示如： { "\_id" : ObjectId("4c479885089df9b53474170a"), "age" : 15, "name" : "user1" }

db.users.update({'name':"user1"}, {'$push':{'data':1}})

for u in db.users.find({'name':"user1"}): print u

# 显示如： { "\_id" : ObjectId("4c479885089df9b53474170a"), "age" : 15, "data" : [ 1 ], "name" : "user1" }

db.users.update({'name':"user1"}, {'$pushAll':{'data':[2,3,4,5]}})

for u in db.users.find({'name':"user1"}): print u

# 显示如： { "\_id" : ObjectId("4c479885089df9b53474170a"), "age" : 15, "data" : [ 1, 2, 3, 4, 5 ], "name" : "user1" }

(7) $addToSet: 和 $push 类似，不过仅在该元素不存在时才添加 (Set 表示不重复元素集合)。

db.users.update({'name':"user2"}, {'$unset':{'data':1}})

db.users.update({'name':"user2"}, {'$addToSet':{'data':1}})

db.users.update({'name':"user2"}, {'$addToSet':{'data':1}})

for u in db.users.find({'name':"user2"}): print u

# 显示： { "\_id" : ObjectId("4c479896089df9b53474170b"), "data" : [ 1 ], "name" : "user2" }

db.users.update({'name':"user2"}, {'$push':{'data':1}})

for u in db.users.find({'name':"user2"}): print u

# 显示： { "\_id" : ObjectId("4c479896089df9b53474170b"), "data" : [ 1, 1 ], "name" : "user2" }

要添加多个元素，使用 $each。

db.users.update({'name':"user2"}, {'$addToSet':{'data':{'$each':[1,2,3,4]}}})

for u in db.users.find({'name':"user2"}): print u

# 显示： {u'age': 12, u'\_id': ObjectId('4c479896089df9b53474170b'), u'data': [1, 1, 2, 3, 4], u'name': u'user2'}

# 貌似不会自动删除重复

(8) $each 添加多个元素用。

db.users.update({'name':"user2"}, {'$unset':{'data':1}})

db.users.update({'name':"user2"}, {'$addToSet':{'data':1}})

for u in db.users.find({'name':"user2"}): print u

# 显示： { "\_id" : ObjectId("4c479896089df9b53474170b"), "data" : [ 1 ], "name" : "user2" }

db.users.update({'name':"user2"}, {'$addToSet':{'data':{'$each':[1,2,3,4]}}})

for u in db.users.find({'name':"user2"}): print u

# 显示： {u'age': 12, u'\_id': ObjectId('4c479896089df9b53474170b'), u'data': [1, 2, 3, 4], u'name': u'user2'}

db.users.update({'name':"user2"}, {'$addToSet':{'data':[1,2,3,4]}})

for u in db.users.find({'name':"user2"}): print u

# 显示： { "\_id" : ObjectId("4c479896089df9b53474170b"), "data" : [ 1, 2, 3, 4, [ 1, 2, 3, 4 ] ], "name" : "user2" }

db.users.update({'name':"user2"}, {'$unset':{'data':1}})

db.users.update({'name':"user2"}, {'$addToSet':{'data':[1,2,3,4]}})

for u in db.users.find({'name':"user2"}): print u

# 显示： { "\_id" : ObjectId("4c47a133b48cde79c6780df0"), "data" : [ [1, 2, 3, 4] ], "name" : "user2" }

(9) $pop: 移除数组属性的元素(按数组下标移除)，$pull 按值移除，$pullAll 移除所有符合提交的元素。

db.users.update({'name':"user2"}, {'$unset':{'data':1}})

db.users.update({'name':"user2"}, {'$addToSet':{'data':{'$each':[1, 2, 3, 4, 5, 6, 7, 2, 3 ]}}})

for u in db.users.find({'name':"user2"}): print u

# 显示： { "\_id" : ObjectId("4c47a133b48cde79c6780df0"), "data" : [ 1, 2, 3, 4, 5, 6, 7, 2, 3 ], "name" : "user2" }

db.users.update({'name':"user2"}, {'$pop':{'data':1}}) # 移除最后一个元素

for u in db.users.find({'name':"user2"}): print u

# 显示： { "\_id" : ObjectId("4c47a133b48cde79c6780df0"), "data" : [ 1, 2, 3, 4, 5, 6, 7, 2 ], "name" : "user2" }

db.users.update({'name':"user2"}, {'$pop':{'data':-1}}) # 移除第一个元素

for u in db.users.find({'name':"user2"}): print u

# 显示： { "\_id" : ObjectId("4c47a133b48cde79c6780df0"), "data" : [ 2, 3, 4, 5, 6, 7, 2 ], "name" : "user2" }

db.users.update({'name':"user2"}, {'$pull':{'data':2}}) # 移除全部 2

for u in db.users.find({'name':"user2"}): print u

# 显示： { "\_id" : ObjectId("4c47a133b48cde79c6780df0"), "data" : [ 3, 4, 5, 6, 7 ], "name" : "user2" }

db.users.update({'name':"user2"}, {'$pullAll':{'data':[3,5,6]}}) # 移除 3,5,6

for u in db.users.find({'name':"user2"}): print u

# 显示： { "\_id" : ObjectId("4c47a133b48cde79c6780df0"), "data" : [ 4, 7 ], "name" : "user2" }

(10) $where: 用 JS 代码来代替有些丑陋的 $lt、$gt。

MongoDB 内置了 Javascript Engine (SpiderMonkey)。可直接使用 JS Expression，甚至使用 JS Function 写更复杂的 Code Block。

db.users.remove() # 删除集合里的所有记录

for i in range(10):

db.users.insert({'name':"user" + str(i), 'age':i})

for u in db.users.find(): print u

# 显示如下：

{ "\_id" : ObjectId("4c47b3372a9b2be866da226e"), "name" : "user0", "age" : 0 }

{ "\_id" : ObjectId("4c47b3372a9b2be866da226f"), "name" : "user1", "age" : 1 }

{ "\_id" : ObjectId("4c47b3372a9b2be866da2270"), "name" : "user2", "age" : 2 }

{ "\_id" : ObjectId("4c47b3372a9b2be866da2271"), "name" : "user3", "age" : 3 }

{ "\_id" : ObjectId("4c47b3372a9b2be866da2272"), "name" : "user4", "age" : 4 }

{ "\_id" : ObjectId("4c47b3372a9b2be866da2273"), "name" : "user5", "age" : 5 }

{ "\_id" : ObjectId("4c47b3372a9b2be866da2274"), "name" : "user6", "age" : 6 }

{ "\_id" : ObjectId("4c47b3372a9b2be866da2275"), "name" : "user7", "age" : 7 }

{ "\_id" : ObjectId("4c47b3372a9b2be866da2276"), "name" : "user8", "age" : 8 }

{ "\_id" : ObjectId("4c47b3372a9b2be866da2277"), "name" : "user9", "age" : 9 }

for u in db.users.find({"$where":"this.age > 7 || this.age < 3"}): print u

# 显示如下：

{u'age': 0.0, u'\_id': ObjectId('4c47b3372a9b2be866da226e'), u'name': u'user0'}

{u'age': 1.0, u'\_id': ObjectId('4c47b3372a9b2be866da226f'), u'name': u'user1'}

{u'age': 2.0, u'\_id': ObjectId('4c47b3372a9b2be866da2270'), u'name': u'user2'}

{u'age': 8.0, u'\_id': ObjectId('4c47b3372a9b2be866da2276'), u'name': u'user8'}

{u'age': 9.0, u'\_id': ObjectId('4c47b3372a9b2be866da2277'), u'name': u'user9'}

for u in db.users.find().where("this.age > 7 || this.age < 3"): print u

# 显示如下：

{u'age': 0.0, u'\_id': ObjectId('4c47b3372a9b2be866da226e'), u'name': u'user0'}

{u'age': 1.0, u'\_id': ObjectId('4c47b3372a9b2be866da226f'), u'name': u'user1'}

{u'age': 2.0, u'\_id': ObjectId('4c47b3372a9b2be866da2270'), u'name': u'user2'}

{u'age': 8.0, u'\_id': ObjectId('4c47b3372a9b2be866da2276'), u'name': u'user8'}

{u'age': 9.0, u'\_id': ObjectId('4c47b3372a9b2be866da2277'), u'name': u'user9'}

# 使用自定义的 function, javascript语法的

for u in db.users.find().where("function() { return this.age > 7 || this.age < 3;}"): print u

# 显示如下：

{u'age': 0.0, u'\_id': ObjectId('4c47b3372a9b2be866da226e'), u'name': u'user0'}

{u'age': 1.0, u'\_id': ObjectId('4c47b3372a9b2be866da226f'), u'name': u'user1'}

{u'age': 2.0, u'\_id': ObjectId('4c47b3372a9b2be866da2270'), u'name': u'user2'}

{u'age': 8.0, u'\_id': ObjectId('4c47b3372a9b2be866da2276'), u'name': u'user8'}

{u'age': 9.0, u'\_id': ObjectId('4c47b3372a9b2be866da2277'), u'name': u'user9'}