NoSQL

NoSQL是对非SQL、非传统关系型数据库的统称。
NoSQL一词诞生于1998年，2009年这个词汇被再次提出指非关系型、分布式、不提供ACID的数据库设计模式。

随着互联网时代的到来，数据爆发式增长，数据库技术发展日新月异，要适应新的业务需求。
随着移动互联网、物联网的到来，大数据的技术中NoSQL也同样重要。

https://db-engines.com/en/ranking

分类

- Key-value Store
  - redis、memcached
- Document Store
  - mongodb、CouchDB
- Column Store列存数据库，Column-Oriented DB
  - HBase、Cassandra
- Graph DB
  - Neo4j
- Time Series 时序数据库
  - InfluxDB

Rank Aug 2021	Rank Jul 2021	Rank Aug 2020	DBMS	Database Model	Score Aug 2021	Score Jul 2021	Score Aug 2020
1.	1.	1.	Oracle	Relational, Multi-model	1269.26	+6.59	-85.90
2.	2.	2.	MySQL	Relational, Multi-model	1238.22	+9.84	-23.36
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	973.35	-8.61	-102.53
4.	4.	4.	PostgreSQL	Relational, Multi-model	577.05	-0.10	+40.28
5.	5.	5.	MongoDB	Document, Multi-model	496.54	+0.38	+52.98
6.	6.	↑ 7.	Redis	Key-value, Multi-model	169.88	+1.58	+17.01
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model	165.46	+0.31	+3.01
8.	8.	8.	Elasticsearch	Search engine, Multi-model	157.08	+1.32	+4.76
9.	9.	9.	SQLite	Relational	129.81	-0.39	+3.00
10.	↑ 11.	10.	Microsoft Access	Relational	114.84	+1.39	-5.02
11.	↓ 10.	11.	Cassandra	Wide column	113.66	-0.35	-6.18
12.	12.	12.	MariaDB	Relational, Multi-model	98.98	+0.99	+8.06
13.	13.	13.	Splunk	Search engine	90.60	+0.55	+0.69
14.	14.	↑ 15.	Hive	Relational	83.93	+1.26	+8.64
15.	15.	↑ 17.	Microsoft Azure SQL Database	Relational, Multi-model	75.15	-0.06	+18.31
16.	16.	16.	Amazon DynamoDB	Multi-model	74.90	-0.30	+10.15
17.	17.	↓ 14.	Teradata	Relational, Multi-model	68.82	-0.13	-7.96
18.	18.	↑ 21.	Neo4j	Graph	56.95	-0.21	+6.77
19.	19.	19.	SAP HANA	Relational, Multi-model	55.57	+1.76	+2.46
20.	20.	20.	Solr	Search engine, Multi-model	51.06	-0.73	-0.63
21.	21.	↑ 23.	FileMaker	Relational	50.28	-0.01	+2.24
22.	22.	↓ 18.	SAP Adaptive Server	Relational, Multi-model	47.61	-0.04	-6.35
23.	↑ 25.	↑ 107.	Snowflake	Relational	46.54	+6.50	+43.95
24.	↓ 23.	↓ 22.	HBase	Wide column	44.64	+0.58	-4.47

MongoDB

它是由C++编写的分布式文档数据库。

内部使用类似于Json的bson格式。

中文手册 https://www.w3cschool.cn/mongodb/

官方文档 https://docs.mongodb.com/manual/

安装

5.x已经发布，但是还处于Bug修复版本，不能用在生产环境中。

下载社区版 https://www.mongodb.com/download-center/community

https://www.mongodb.com/download-center/community/releases

windows下载官方zip，解压即可使用。

CentOS8现在对应rpm包安装。`/usr/lib/systemd/system/mongod.service`

组件	文件名
Server	mongod.exe
Router	mongos.exe, Query Router, Sharding Cluster
Client	mongo.exe
MonitoringTools	mongostat.exe,mongotop.exe
ImportExportTools	mongodump.exe, mongorestore.exe, mongoexport.exe, mongoimport.exe
MiscellaneousTools	bsondump.exe, mongofiles.exe, mongooplog.exe, mongoperf.exe

CentOS下载运行

```
$ yum install mongodb-org-server-5.0.2-1.el8.x86_64.rpm

$ systemctl enable mongod
$ systemctl start mongod
```

Windows运行

mongod.exe是Server，mongo.exe是Client。

```
$ cd  /o/mongodb3.6/bin
$ ./mongod.exe

运行报错了，如下，不同版本返回略有不同
2021-08-02T14:26:13.234-0700 I STORAGE  [initandlisten] exception in initAndListen: NonExistentPath: Data directory O:\data\db\ not found., terminating

{"t":{"$date":"2021-09-01T15:48:18.043+08:00"},"s":"E",  "c":"CONTROL",  "id":20557,   "ctx":"initandlisten","msg":"DBException in initAndListen, terminating","attr":{"error":"NonExistentPath: Data directory O:\\data\\db\\ not found. Create the missing directory or specify another path using (1) the --dbpath command line option, or (2) by adding the 'storage.dbPath' option in the configuration file."}}

启动服务出错，原因在于找不到数据目录。默认是/data/db
windows下在当前盘符根目录下创建目录即可`o:/data/db`
可以使用--dbpath指定数据存储路径
```

选项说明

- --bind_ip ip 逗号分隔IP地址。默认localhost
- --bind_ip_all 绑定所有本地IP地址
- --port port 端口，默认27017
- --dbpath path 数据路径，缺省为`\data\db\`。windows下缺省就是当前盘符的根目录
- --logpath path 指定日志文件，替代stdout，说明默认是控制台打印日志
- -f file 指定配置文件，yaml格式
- 注册windows服务
  - --install 注册windows服务
  - --serviceName name 服务名称
  - --serviceDisplayName name 服务显示名

Windows下注册为服务的命令如下，使用了配置文件：

`$ mongod.exe -f "o:/mongodb3.6/bin/mongod.yml" --serviceName mongod --serviceDisplayName mongo --install`

注意，注册服务得需要管理员权限。

配置

配置文件`/etc/mongod.conf`，采用YAML格式。

YAML格式，嵌套使用缩进完成，不支持Tab等制表符，支持空格。冒号之后要有空格。

YAML参考 https://www.w3cschool.cn/iqmrhf/dotvpozt.html

配置参考 http://mongoing.com/docs/reference/configuration-options.html

```
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log
```

```
# Where and how to store data.
storage:
  dbPath: /var/lib/mongo
  journal:
    enabled: true

# how the process runs
processManagement:
  fork: true  # fork and run in background
  pidFilePath: /var/run/mongodb/mongod.pid  # location of pidfile
  timeZoneInfo: /usr/share/zoneinfo

# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1
```

选项

- systemLog
  - destination，缺省是输出日志到std，file表示输出到文件
  - path，日志文件路径。文件目录必须存在
  - logAppend，true表示在已存在的日志文件追加。默认false，每次启动服务，重新创建新的日志。
- storage
  - dbPath，必须指定mongodb的数据目录，**目录必须存在**
  - journal，64位系统默认为数据存储开启日志
- net
  - bindIp，缺省绑定到127.0.0.1
  - port，端口，缺省为27017，客户端连接用

dbPath为`/var/lib/mongo`

windows简单配置

```
storage:
  dbPath: /mongodb-win32-x86_64-windows-5.0.2/db
  journal:
    enabled: true
net:
  bindIp: 127.0.0.1
  port: 27017
```

```
bin/mongod.exe --config ./mongo.yml
```

客户端

客户端连接

CentOS还需要下载mongodb-org-shell-5.0.2-1.el8.x86_64.rpm，安装客户端。

```
$ mongo
> help
> show dbs            查看当前有哪些库
> use mytest          use库，若库不存在，库不会立即创建，除非写入数据
> db                  显示当前使用库
> db.users.insert({"user":"tom", "age":20})
WriteResult({ "nInserted" : 1 })
```

`db.users.insert`为当前db的users集合插入`文档`，同时创建了`库`mytest，创建了`集合`users。

也可以使用官方的可视化工具Compass。https://www.mongodb.com/products/compass

Python连接

Mongodb官方推荐使用pymongo。参看 https://docs.mongodb.com/drivers/pymongo/

缺省安装pymongo 3.x，支持MongoDB2.6+，兼容Python 3.4+。

```
$ pip install pymongo
```

连接字符串

```
mongodb://[username:password@]host1[:port1][,...hostN[:portN]]
[/[defaultauthdb][?options]]

mongodb://username:password@127.0.0.1:27017/test
```

```
import pymongo

client = pymongo.MongoClient('192.168.142.130', 27017)
print(client)

db = client.mytest # 什么原理？既能用属性访问？又能像key一样访问
# db = client['mytest'] # 指定数据库 # Database类
print(type(db), db)

users = db.users # 集合 # Collection类
# users = db['users']
print(users)

print(*users.find()) # 返回结果，可迭代对象

client.close()
```

既能用属性访问，又能像key一样访问，一定用到了魔术方法的`__getattr__`、`__getitem__`。

基本概念

MongoDB中可以创建使用多个库，但有一些数据库名是保留的，可以直接访问这些有特殊作用的数据库。

- admin： 从权限的角度来看，这是"root"数据库。要是将一个用户添加到这个数据库，这个用户自动继承所有数据库的权限。一些特定的服务器端命令也只能从这个数据库运行，比如列出所有的数据库或者关闭服务器。
- local: 这个数据永远不会被复制，可以用来存储限于本地单台服务器的任意集合
- config: 当Mongo用于分片设置时，config数据库在内部使用，用于保存分片的相关信息。

RDBMS	MongoDB
Database	Database
Table	Collection
Row	Document
Column	Field
Join	Embedded Document嵌入文档或Reference引用
Primary Key	主键 (MongoDB提供了key为 _id)

插入数据

```python
from pymongo.results import InsertOneResult

user1 = {'id':'1', 'name':'ben', 'age':20}
# users.insert(user1) # 已经过期
# 单条插入
x:InsertOneResult = users.insert_one(user1)
print(type(x), x)
print(x.inserted_id) # 5d45546f42a331914ccaaf06

user2 = {'id':257, 'name':'tom', 'age':32}
user3 = {'id':258, 'name':'jerry', 'age':48}

# 批量插入
result = users.insert_many([user2, user3])
print(result.inserted_ids)
# [ObjectId('5d4557be12f2bf1448c743de'),
#  ObjectId('5d4557be12f2bf1448c743df')]

# 大小写敏感
user4 = {'id':'3', 'name':'tom', 'age':20, 'Name':'tommy'}
x = users.insert_one(user4)
print(x.inserted_id)
```

每条数据插入后都有一个唯一key，属性`_id`唯一标识一个文档。没有没有显式指明该属性，会自动生成一个ObjectId类型的`_id`属性。

ObjectId由12字节组成

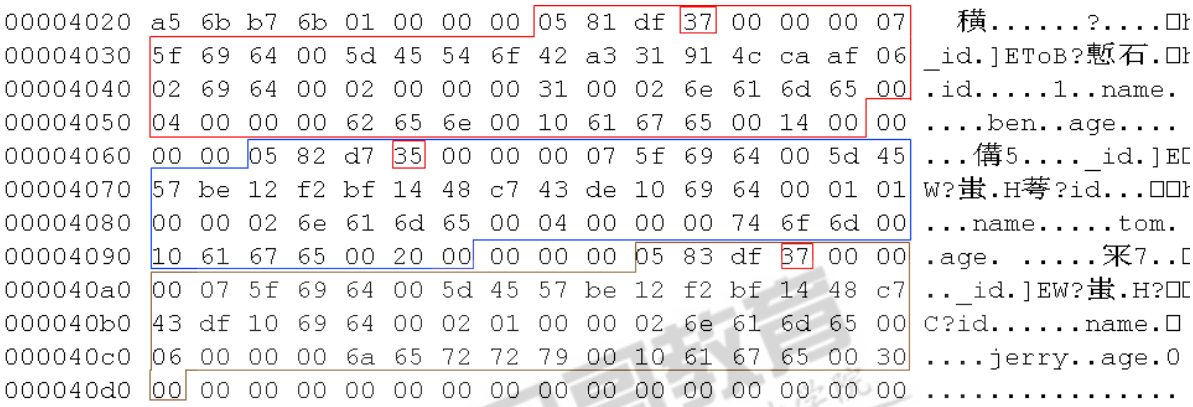
- 4字节时间戳
- 3字节机器识别码
- 2字节进程id
- 3字节随机数

```
import bson
print(bson.ObjectId('617fb807a9620927b0c08e90').generation_time)
```

文档

每一条记录对应一个文档，其格式使用BSON。BSON即Binary JSON。

BSON二进制格式如下




```
00 00 05 82 d7 35 00 00 00 07 5f 69 64 00 5d 45
35 表示文档二进制数据长度
07 表示MongoDB中特殊数据类型ObjectId
5f 69 64 ，ascii码_id

57 be 12 f2 bf 14 48 c7 43 de 10 69 64 00 01 01
5d 45 57 be 12 f2 bf 14 48 c7 43 de ，
ObjectId('5d4557be12f2bf1448c743de')_id的值
10 69 64 ，10表示类型int-32，69 64是ascii码id
0101 小端模式，十进制257

00 00 02 6e 61 6d 65 00 04 00 00 00 74 6f 6d 00
02，表示数据类型UTF-8 String
6e 61 6d 65表示字符串name
04，表示字符串长度
74 6f 6d 00，表示字符串tom和结束符

10 61 67 65 00 20 00
10 ，表示数据类型int-32
61 67 65，表示字符串age
20，表示十进制32
```

类型参考 https://docs.mongodb.com/v3.6/reference/bson-types/

文档

- 文档中，使用键值对
- 文档中的键/值对是**有序**的

- 键是字符串
  - **区分大小写**，使用UTF-8字符
  - 键不能含有\0 (空字符)。这个字符用来表示键字符串的结尾
    `.`和`$`有特别的意义，只有在特定环境下才能使用
    以下划线`_`开头的键是保留的，例如`_id`
- 值可以是
  - 字符串、32位或64位整数、双精度、时间戳（毫秒）、布尔型、null
  - 字节数组、BSON数组、BSON对象

查询

Pymongo中`{'name':'tom'}`称为filter。

单条查询

find_one第一参数是filter，相当于SQL的where子句。

```
# 查询
result = users.find_one({'name':'tom'})
print(type(result), result) # 返回字典带_id
# 使用key查询
from bson.objectid import ObjectId
result = users.find_one({'_id':ObjectId('5d48d95d4fd47046028b8e54')})
print(type(result), result)

result = users.find_one({'name':'tommy'})
print(type(result), result) # 查不到，返回None
```

多条查询

```
# 多条查询
results = users.find({'name':'tom'})
print(type(results)) # pymongo.cursor.Cursor对象
print(results) # 可迭代对象

for x in results:
    print(type(x), x) # 字典
```

返回游标，可迭代对象

查询操作

比较符号	含义	示例
$lt	小于	`{'age': {'$lt': 20}}`
$gt	大于	`{'age': {'$gt': 20}}`
$lte	小于等于	`{'age': {'$lte': 20}}`
$gte	大于等于	`{'age': {'$gte': 20}}`
$ne	不等于	`{'age': {'$ne': 20}}`
$eq	等于，可以不用这个符号	`{'age': {'$eq': 20}}`
$in	在范围内	`{'age': {'$in': [20, 23]}}`
$nin	不在范围内	`{'age': {'$nin': [20, 23]}}`

逻辑符号	含义	示例
$and	与	`{'$and':[{'name':'tom'}, {'age':{'$gt':20}}]}` `{'name':'tom', 'age':{'$gt':20}}`
$or	或	`{'$or':[{'name':'tom'}, {'age':{'$gt':20}}]}`
$not	非	`{'age':{'$not':{'$lte':20}}`

元素	含义	示例
$exists	文档中是否有这个字段	`{'Name':{'$exists':True}}`
$type	字段是否是指定的类型	`{'age':{'$type':16}}`

常用类型

- 字符串类型编码为2，别名为string
- 整型编码为16，别名为int
- 长整型编码为18，别名为long

`https://docs.mongodb.com/manual/reference/operator/query/type/#op._S_type`

操作符	含义	示例
$regex	文档中字段内容匹配	`{'name':{'$regex':'^t'}}` `{'name':{'$regex':'m$'}}`
$mod	取模	`{'age':{'$mod':[10, 2]}}` age值能模10余2的

投影

```
results = users.find({'age':{'$gt':20}}, ('name', 'id')) # 允许
print(*results)

results = users.find({'age':{'$gt':20}}, {'name':0, 'age': False}) # 排除
print(*results)

result = table.aggregate([
{'$project':{'name':1}}
])
print(*result, sep='\n')
print('=' * 30)

print(*table.find({}, ('name',)), sep='\n')
```

可以使用列表、元组、字典描述投影字段，或排除投影的字段。

统计

```
count = users.find({'age':{'$gt':20}}).count() # 过期了
print(count)
print(users.count_documents({'age':{'$gt':20}}))

age = users.find().distinct('age')
print(age)
```

排序

```
results = users.find().sort('name', pymongo.DESCENDING)
print(*results, sep='\n')
print('-' * 30)

results = users.find().sort([
    ('name', pymongo.DESCENDING),
    ('age', pymongo.ASCENDING)
])
print(*results, sep='\n')
```

分页

```
results = users.find()
print(*results, sep='\n')
print('-' * 30)

results = users.find().skip(2)
print(*results, sep='\n')
print('-' * 30)

results = users.find().skip(1).limit(2)
print(*results, sep='\n')
print('-' * 30)
```

skip 跳过几个，limit限制结果个数。

聚合

MongoDB实现聚合流水线。用多个构件创建一个管道（pipeline），用于对一连串的文档的处理。

可以理解为对一个文档进行逐级处理，一环扣一环，形成一条处理流程链，前一级输出作为下一级输入。

每一级称为一个stage阶段。

stage参考 https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/

```
print(*users.find({}, projection=('name',)))
print(*users.find({}, ('name',)))
print('~' * 30)

print(*users.aggregate([
    {'$project':{'name':1}}, # 流水线第一级，投影stage
]))
如果简单投影来说，还是上面的方便
```

```
print(*users.aggregate([
    {'$project':{'name':1}}, # 流水线第一级
    {'$count': 'c'}, # 流水线第二级，统计上一级传入的条目数
]))
```

`$count`是pipeline的stage指令。

`$group`，对所有输入文档进行分组，对指定的字段进行聚合运算，输出仅包含指定字段和聚合结果。

```
{
  '$group':
    {
      '_id': <expression>, // 分组表达式，指定分组字段的
      '<field1>': { <accumulator1> : <expression1> }, //
      ...
    }
 }


```

```
_id     分组字段，如果要用字段的值，一定之前加上$
field1

$age是引用age字段的值
```

不指定分组

```
print(*users.aggregate([
    {
        '$group':{
            '_id': None, # 不指定分组，就看做一组
            'c0':{'$count': {}},  # $count没参数用{}
            'c1':{'$sum':1},      # 使用1累计
        }
    }
]))
{'_id': None, 'c0': 5, 'c1': 5}

print(*users.aggregate([
    {
        '$group':{
            '_id': None, # 不指定分组，就看做一组
            'c0':{'$count': {}},  # $count没参数用{}
            'c1':{'$sum':1},      # 使用1累计
            'c2':{'$sum':'$age'}, # $sum后面指定统计的字段
            'c3':{'$avg':'$age'}, # age前一定要加$，表示使用age字段值
            'c4':{'$max':'$age'},
            'c5':{'$min':'$age'}
        }
    }
]))
{'_id': None, 'c0': 5, 'c1': 5, 'c2': 184.0, 'c3': 36.8, 'c4': 64, 'c5':
20.0}
```

`$count`是累加器操作符。它不是聚合的stage指令。

指定分组

```
print(*users.aggregate([
    {
        '$group':{
            '_id': '$name', # 按照name字段值分组
            'c0':{'$count': {}},  # $count没参数用{}
            'c1':{'$sum':1},      # 使用1累计
            'c2':{'$sum':'$age'}, # $sum后面指定统计的字段
            'c3':{'$avg':'$age'}, # age前一定要加$，表示使用age字段值
            'c4':{'$max':'$age'},
            'c5':{'$min':'$age'}
        }
    }
]), sep='\n')

{'_id': 'tom', 'c0': 2, 'c1': 2, 'c2': 52.0, 'c3': 26.0, 'c4': 32, 'c5':
20.0}
{'_id': 'jerry', 'c0': 1, 'c1': 1, 'c2': 48, 'c3': 48.0, 'c4': 48, 'c5': 48}
{'_id': 'ben', 'c0': 1, 'c1': 1, 'c2': 20, 'c3': 20.0, 'c4': 20, 'c5': 20}
```

```
{'_id': 'sam', 'c0': 1, 'c1': 1, 'c2': 64, 'c3': 64.0, 'c4': 64, 'c5': 64}
```

对流水线指定条件

```
print(*users.aggregate([
    {'$match':{'age':{'$lt':40}}}, # 第一级，$match指定filter过滤数据
    {   # 第二级，分组
        '$group':{
            '_id': '$name', # 按照name字段值分组
            'c0':{'$count': {}},  # $count没参数用{}
            'c1':{'$sum':1},      # 使用1累计
            'c2':{'$sum':'$age'}, # $sum后面指定统计的字段
            'c3':{'$avg':'$age'}, # age前一定要加$，表示使用age字段值
            'c4':{'$max':'$age'},
            'c5':{'$min':'$age'}
        }
    }
]), sep='\n')

{'_id': 'tom', 'c0': 2, 'c1': 2, 'c2': 52.0, 'c3': 26.0, 'c4': 32, 'c5': 20.0}
{'_id': 'ben', 'c0': 1, 'c1': 1, 'c2': 20, 'c3': 20.0, 'c4': 20, 'c5': 20}
```

更新

更新操作符	含义	示例
$inc	对给定字段数字值增减	`{'$inc':{'age':-5}}`
$set	设置字段值，如果字段不存在则创建	`{'$set':{'gender':'M'}}`
$unset	移除字段	`{'$unset':{'Name':""}}`

update_one只更新第一个

```
import pymongo
from pymongo.collection import Collection
from pymongo.results import InsertOneResult, UpdateResult, DeleteResult, InsertManyResult


client = pymongo.MongoClient()
db = client.mytest
users:Collection = db.users

filter = {'name':'tom'}
print(*users.find(filter))

x:UpdateResult = users.update_one(filter, {'$inc':{'age': 5}})
print(x.matched_count, x.modified_count)

print(*users.find(filter))
```

```
只有一条文档的age增加了5。减5就使用-5
{'_id': ObjectId('617fa9bdf574bc11dbb86792'), 'name': 'tom', 'age': 25.0}
{'_id': ObjectId('617fb807a9620927b0c08e8e'), 'id': 257, 'name': 'tom',
'age': 32}
1 1
{'_id': ObjectId('617fa9bdf574bc11dbb86792'), 'name': 'tom', 'age': 30.0}
{'_id': ObjectId('617fb807a9620927b0c08e8e'), 'id': 257, 'name': 'tom',
'age': 32}
```

update_many更新多行

```
filter = {'name':'tom'}
print(*users.find(filter))

x:UpdateResult = users.update_many(filter, {'$set':{'gender': 'M'}})
print(x.matched_count, x.modified_count)

print(*users.find(filter))
```

```
filter = {}
print(*users.find(filter))

x:UpdateResult = users.update_many(filter, {'$unset':{'Name': ''}})  # 删除字
段Name
print(x.matched_count, x.modified_count)

print(*users.find(filter))
```

replace_one替换一个文档

更新除_id字段外的所有字段，这就是更新一个记录了，这个记录在MongoDB中就是文档

```
filter = {'name':'tom'}
print(*users.find(filter))

x:UpdateResult = users.replace_one(filter, {'name':'tom', 'age':36, 'id':10})
print(x.matched_count, x.modified_count)

print(*users.find(filter))
```

```
{'_id': ObjectId('617fa9bdf574bc11dbb86792'), 'name': 'tom', 'age': 30.0,
'gender': 'M'} {'_id': ObjectId('617fb807a9620927b0c08e8e'), 'id': 257,
'name': 'tom', 'age': 32, 'gender': 'M'}
1 1
{'_id': ObjectId('617fa9bdf574bc11dbb86792'), 'name': 'tom', 'age': 36, 'id':
10} {'_id': ObjectId('617fb807a9620927b0c08e8e'), 'id': 257, 'name': 'tom',
'age': 32, 'gender': 'M'}
```

删除

```python
# 删除
# users.remove({'Name':'tommy'}) # 已经过期
import pymongo
from pymongo.collection import Collection
from pymongo.results import InsertOneResult, UpdateResult, DeleteResult,
InsertManyResult


client = pymongo.MongoClient()
db = client.mytest
users:Collection = db.users

filter = {'name':'tom'}
print(*users.find(filter))
x:DeleteResult = users.delete_one(filter)
print(x.deleted_count)
print(*users.find(filter))

filter = {'gender':{'$exists': True}}
x:DeleteResult = users.delete_many(filter)
print(x.deleted_count)
```

`db.collection.delete_many({})` 删除所有文档，**慎用**

全文索引

https://docs.mongodb.com/manual/text-search/

MongoDB会在_id字段上创建唯一键索引。

在使用全文搜索的集合上必须建立一个文本搜索索引，只能有一个，可以多个字段一起。

```python
r = users.create_index([('name', 'text'),('comment','text')]) # 字典传参在
pymongo中出错，使用可迭代二元组
print(type(r), r)
```

$text运算符进行文本搜索，提供的搜索字符串，使用空格或标点分割多个词，表示**或**关系。

```python
r = users.find({'$text':{'$search':'china tom'}})
print(*r, sep='\n')
```