1. NoSQL简介

1.1. NoSQL的概念

```
    NoSQL(NoSQL = Not Only SQL)，意即“不仅仅是SQL”，是一项全新的数据库革命性运动。
    NoSQL一词最早出现于1998年，是Carlo Strozzi开发的一个轻量、开源、不提供SQL功能的关系数据库.
    2009年，Last.fm的Johan Oskarsson发起了一次关于分布式开源数据库的讨论，来自Rackspace的Eric Evans再次提出
NoSQL的概念，这时的NoSQL主要指非关系型、分布式、不提供ACID的数据库设计模式。
    2009年在亚特兰大举行的"no:sql(east)"讨论会是一个里程碑，其口号是
    "select fun, profit from real_world where relational=false;"。
因此，对NoSQL最普遍的解释是“非关系型的”，强调Key-Value Stores和文档数据库的优点，而不是单纯的反对RDBMS。
NoSQL的拥护者们提倡运用非关系型的数据存储，相对于铺天盖地的关系型数据库运用，这一概念无疑是一种全新的思维的注入。
```

1.2. NoSQL的适用场合

```
NoSQL数据库在以下的这几种情况下比较适用：
1、数据模型比较简单；
2、需要灵活性更强的IT系统；
3、对数据库性能要求较高；
4、不需要高度的数据一致性；
5、对于给定key，比较容易映射复杂值的环境。

网站数据：     MongoDB 非常适合实时的插入，更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性
缓存：        由于性能很高，MongoDB 也适合作为信息基础设施的缓存层。在系统重启之后，
              由 MongoDB 搭建的持久化缓存层可以避免下层的数据源过载
              大尺寸，低价值的数据：使用传统的关系型数据库存储一些数据时可能会比较昂贵，在
              此之前，很多时候程序员往往会选择传统的文件进行存储
高伸缩性的场景：
              MongoDB 非常适合由数十或数百台服务器组成的数据库。
              MongoDB 中已经包含对 MapReduce 引擎的内置支持 用于对象及 JSON 数据的存储：
              MongoDB 的 BSON 数据格式非常适合文档化格式的存储及查询
```

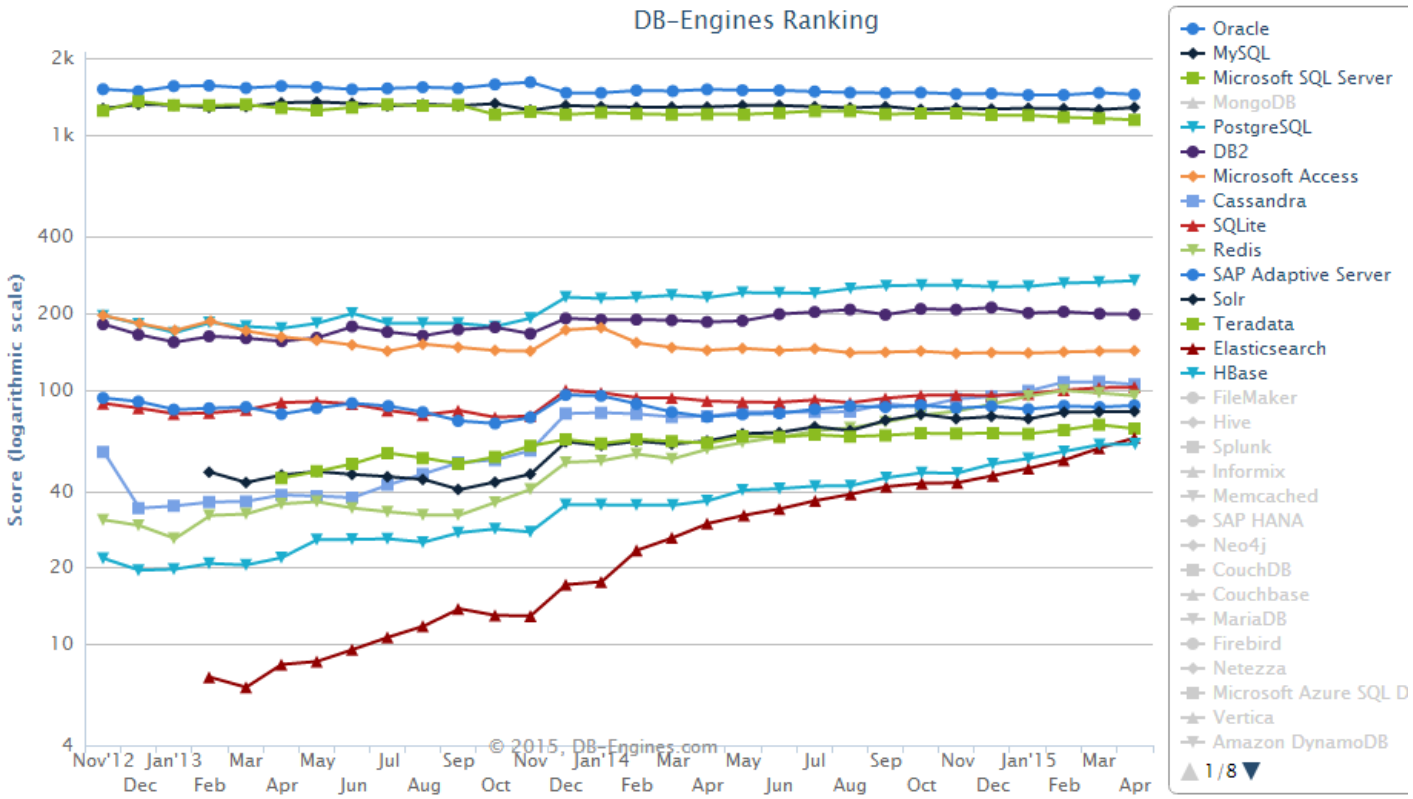
2. MongoDB

2.1. MongoDB是什么

```
MongoDB是由C++语言编写的开源的文档型数据库系统。MongoDB旨在为WEB应用提供可扩展的高性能,高可用数据存储解决方案。
MongoDB is an open-source document database that provides high performance, high availability,\
and automatic scaling.
```

MongoDB被db-engines网站评为2014 年年度最受欢迎数据库管理系统.

下图为著名数据库引擎的发展情况　网址http://db-engines.com/en/ranking_trend



2.1.1. Document Database文档型数据库

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents
文档就是存储在在mongoDB中的一条记录,是一个由键值对组成的数据结构.类似于JSON的对象,键所对应的值也可以包含其他对象,数组,或者一个文档的数组.MongoDB使用了BSON(类似JSON)这种结构来存储数据和网络数据交换.

2.1.2. BSON

```
BSON（Binary Serialized Document Format）是一种类json的一种二进制形式的存储格式，简称Binary JSON.
它和JSON一样，支持内嵌的文档对象和数组对象，但是BSON有JSON没有的一些数据类型，如Date和BinData类型。
BSON可以做为网络数据交换的一种存储形式.
BSON有三个特点：轻量性、可遍历性、高效性
BSON supports the following data types as values in documents. Each data type has a corresponding
number that can be used with the $type operator to query documents by BSON type.
BSON支持下列在 文档中的 数据类型,每一个数据类型都有一个对应的编号以至于能够使用$type操作符来判断BSON的类型.
Type    Number  Notes
Double      1
String      2
Object      3
Array       4
Binary data 5
Undefined   6   Deprecated.
Object id   7
Boolean     8
Date        9
Null        10
Regular Expression  11
JavaScript  13
Symbol      14
JavaScript (with scope) 15
32-bit integer  16
Timestamp   17
64-bit integer  18
Min key     255 Query with -1.
Max key     127

Comparison/Sort Order
```

```
When comparing values of different BSON types, MongoDB uses the following comparison order,
from lowest to highest:
```

```
MinKey (internal type)
Null
Numbers (ints, longs, doubles)
Symbol, String
Object
Array
BinData
ObjectId
Boolean
Date
Timestamp
Regular Expression
MaxKey (internal type)
```

```
    MongoDB treats some types as equivalent for comparison purposes. For instance, numeric types undergo
    conversion before comparison.Changed in version 3.0.0: Date objects sort before Timestamp objects.
    Previously Date and Timestamp objects sorted together.
    The comparison treats a non-existent field as it would an empty BSON Object. As such, a sort on the
    a field in documents { } and { a: null } would treat the documents as equivalent in sort order.
    With arrays, a less-than comparison or an ascending sort compares the smallest element of arrays,
    and a greater-than comparison or a descending sort compares the largest element of the arrays.
    As such, when comparing a field whose value is a single-element array (e.g. [ 1 ]) with non-array
    fields , the comparison is between 1 and 2. A comparison of an empty array (e.g. [ ]) treats
    the empty array as less than null or a missing field.

    MongoDB sorts BinData in the following order:

    First, the length or size of the data.
    Then, by the BSON one-byte subtype.
    Finally, by the data, performing a byte-by-byte comparison.

    The following sections describe special considerations for particular BSON types.
```

- ObjectId

```
    ObjectIds are: small, likely unique, fast to generate, and ordered. These values consists of
    12-bytes, where the first four bytes are a timestamp that reflect the ObjectId’s creation.
    Refer to the ObjectId documentation for more information.
```

- String

```
BSON strings are UTF-8. In general, drivers for each programming language convert from the
language’s string format to UTF-8 when serializing and deserializing BSON. This makes it possible
to  store most international characters in BSON strings with ease. [1] In addition, MongoDB $regex
queries support UTF-8 in the regex string.

[1] Given strings using UTF-8 character sets, using sort() on strings will be reasonably correct.
However, because internally sort() uses the C++ strcmp api, the sort order may handle some
characters incorrectly.
```

- Timestamps

```
BSON has a special timestamp type for internal MongoDB use and is not associated with the regular
Date type. Timestamp values are a 64 bit value where:

the first 32 bits are a time_t value (seconds since the Unix epoch)the second 32 bits are an
incrementing ordinal for operations within a given second.
Within a single mongod instance, timestamp values are always unique.

In replication, the oplog has a ts field. The values in this field reflect the operation time,
which uses a BSON timestamp value.

NOTE
The BSON timestamp type is for internal MongoDB use. For most cases, in application development,
you will want to use the BSON date type. See Date for more information.If you insert a document
containing an empty BSON timestamp in a top-level field, the MongoDB server will replace that empty
timestamp with the current timestamp value. For example, if you create an insert a document with a
timestamp value, as in the following operation:

var a = new Timestamp();

db.test.insert( { ts: a } );
Then, the db.test.find() operation will return a document that resembles the following:

{ "_id" : ObjectId("542c2b97bac0595474108b48"), "ts" : Timestamp(1412180887, 1) }
    If ts were a field in an embedded document, the server would have left it
as an empty timestamp value.

Changed in version 2.6: Previously, the server would only replace empty
timestamp values in the first two fields, including _id, of an inserted
document. Now MongoDB will replace any top-level field.
```

- Date

```
BSON Date is a 64-bit integer that represents the number of milliseconds since the Unix epoch
(Jan 1, 1970). This results in a representable date range of about 290 million years into
the past and future.
The official BSON specification refers to the BSON Date type as the UTC datetime.
Changed in version 2.0: BSON Date type is signed.Negative values represent  dates before 1970.

EXAMPLE
Construct a Date using the new Date() constructor in the mongo shell:

var mydate1 = new Date()
EXAMPLE
Construct a Date using the ISODate() constructor in the mongo shell:

var mydate2 = ISODate()
EXAMPLE
Return the Date value as string:

mydate1.toString()
EXAMPLE
Return the month portion of the Date value; months are zero-indexed, so
that January is month 0:

mydate1.getMonth()
[2] Prior to version 2.0, Date values were incorrectly interpreted as unsigned integers,
which affected sorts, range queries, and indexes on Date fields. Because indexes are not recreated
when upgrading,please re-index if you created an index on Date values with an earlier version, and
dates before 1970 are relevant to your application.
```

2.2. Mongo版本说明

2.2.1. mongoDB Version numbers

- 2.0.0 : Srelease.
- 2.0.1 : Revision.
- 2.1.0 : Development release for testing only. Includes new features and changes for testing. Interfaces and stability may not be compatible in development releases.
- 2.2.0 : Stable release. This is a culmination of the 2.1.x development series

2.2.2. mongoDB Driver version numbers.

```
If your driver has a version number of 2.9.1,2 is the major version, 9 is minor, and 1 is the patch.
```

2.3. Mongo的应用案例

- 京东,中国著名电商,使用MongoDB存储商品信息,支持比价和关注功能.
- 赶集网,中国著名分类信息网站,使用MongoDB记录pv浏览计数
- 奇虎360,著名病毒软件防护和移动应用平台,使用MongoBD支撑的HULK平台每天接受200亿次的查询.
- 百度云,使用MongoDB管理百度云盘中500亿条关于文件源信息的记录.
- CERN，著名的粒子物理研究所，欧洲核子研究中心大型强子对撞机的数据使用MongoDB
- 纽约时报，领先的在线新闻门户网站之一，使用MongoDB
- sourceforge.net，资源网站查找，创建和发布开源软件免费，使用MongoDB的后端存储 另外淘宝也使用了mongoDB MongoDB的特点: 面向集合存储，易存储对象类型的数据。 模式自由。 支持动态查询。 支持完全索引，包含内部对象。 支持查询。 支持复制和故障恢复。 使用高效的二进制数据存储，包括大型对象（如视频等）。 自动处理碎片，以支持云计算层次的扩展性 支持 RUBY，PYTHON，JAVA，C++，PHP等多种语言。 文件存储格式为BSON（一种JSON的扩展）

2.4. 搭建mongoDB开发环境

2.4.1. 安装mongoDB数据库

安装注意事项:

- 64位的Linux是运行MongoDB的最佳选择.CentOS和Ubuntu可能更加适合初学者上手.
- 操作系统应该选用最新发布的稳定版本.
- 如果服务器需要处理大量数据请不要选用32位系统(限制处理2GB的文件)
- 副本集和mongos进程可以运行在32位系统上,其余的不推荐使用32位系统上.
- 可以在Windows上运行mongos进程,Linux上运行mongods作为分片.
- 文件系统的选择(推荐使用ext4 或者 xfs文件系统 作为数据卷)备份时能进行
- 文件快照的文件系统,但是不推荐使用ext3(对数据文件进行预分配的时候耗时过长)

安装文档网址:

RedHat/CentOS下的安装文档

```
http://docs.mongodb.org/manual/tutorial/install-mongodb-on-red-hat/

# cd /etc/yum.repos.d/
# touch mongodb-org-3.0.repo
# vim mongodb-org-3.0.repo
    [mongodb-org-3.0]
    name=MongoDB Repository
    baseurl=http://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/3.0/x86_64/
    gpgcheck=0
    enabled=1
# vim /etc/sudoers
    test ALL=(ALL:ALL) ALL
#sudo yum install -y mongodb-org
```

```
RHEL6/CentOS6
# iptables -I INPUT -p tcp -m tcp --dport 27017 -j ACCEPT
  //或者在图形界面下 手动添加27017到信任端口 不推荐直接关闭防火墙
# service iptables save  //保存防火墙规则
CentOS7/RHEL7
//C++驱动可以使用Yum源中1.53版本的boost库 已经可以满足要求,而不必手动安装boost
# firewall-cmd --zone=public --add-port=27017/tcp --permanent
# firewall-cmd --zone=public --query-port=27017/tcp
# sudo firewall-cmd --reload
```

Ubuntu下的安装文档

```
http://docs.mongodb.org/manual/tutorial/install-mongodb-on-ubuntu/
```

- 1.Import the public key used by the package management system.

  ```
  #sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
  ```

- 2.Create a list file for MongoDB

  ```
  #echo "deb http://repo.mongodb.org/apt/ubuntu "$(lsb_release -sc) "/mongodb-org/3.0 multiverse"|\
  sudo tee /etc/apt/sources.list.d/mongodb-org-3.0.list
  ```

- 3.Reload local package database

```
#sudo apt-get update
```

- 4.Install the MongoDB packages Install the latest stable version of MongoDB

```
#sudo apt-get install -y mongodb-org
```

Install a specific release of MongoDB

```
#sudo apt-get install -y mongodb-org=3.0.0 mongodb-org-server=3.0.0 mongodb-org-shell=3.0.0\
mongodb-org-mongos=3.0.0 mongodb-org-tools=3.0.0
```

- 5.Run MongoDB

```
1.Start MongoDB.
Issue the following command to start mongod:
    #sudo service mongod start
2.Verify that MongoDB has started successfully
Verify that the mongod process has started successfully by checking
the contents of the log file at /var/log/mongodb/mongod.log for a line reading

[initandlisten] waiting for connections on port <port>
where <port> is the port configured in /etc/mongod.conf, 27017 by default.

3.Stop MongoDB.
As needed, you can stop the mongod process by issuing the following command:

    #sudo service mongod stop
4.Restart MongoDB.
Issue the following command to restart mongod:

    #sudo service mongod restart
5.Begin using MongoDB.
To begin using MongoDB, see Getting Started with MongoDB.
Also consider the Production Notes document before deploying MongoDB in a production environment.
Later, to stop MongoDB, press Control+C in the terminal where the mongod instance is running.
```

2.4.2. 总结:

```
    #sudo service mongod status(查看状态)/restart(重启)/start(启动)/stop(停止)
```

2.5. mongoDB基本操作上手

```
//show dbs                      show database names
//show collections              show collections in current database
//show users                    show users in current database

#sudo service mongod stop
    mongod stop/waiting
#sudo service mongod status
    mongod stop/waiting
#sudo service mongod start
    mongod start/running, process 4469
待mongoDB服务启动后再shell中输入mongo进入mongoshell
#mongo
    MongoDB shell version: 3.0.0
    connecting to: test
    Welcome to the MongoDB shell.
    For interactive help, type "help".
    For more comprehensive documentation, see
    http://docs.mongodb.org/
    Questions? Try the support group
    http://groups.google.com/group/mongodb-user
> show dbs
显示有哪些数据库
    local  0.078GB

> use foolbar
切换到foolbar数据库,如果foolbar数据库不存在,会在该数据第一次插入之后创建
    switched to db foolbar
> db
查看当前数据库的名称
    foolbar
> show collections;
显示当前数据库中有哪些集合
>db.blog.insert({"title1":"instrduce of mongo"});
第一次插入数据会创建数据库(集合) 插入的数据是一个文档
    WriteResult({ "nInserted" : 1 })
> db.blog.find()
查找当前数据库中blog集合的所有文档
    { "_id" : ObjectId("5503ecd83853a67e204962ae"), "title1" : "instrduce of mongo" }
> db.blog.insert({"docment":"hello mongoDB!"});
    WriteResult({ "nInserted" : 1 })
> db.blog.find()
    { "_id" : ObjectId("5503ecd83853a67e204962ae"), "title1" : "instrduce of mongo" }
    { "_id" : ObjectId("5503ed873853a67e204962af"), "docment" : "hello mongoDB!" }
> db.blog.insert({"title" : "mytest", "name" : "pc"});
    WriteResult({ "nInserted" : 1 })
> db.blog.insert({"title" : "mytest", "name" : "pc"});
```

```
    WriteResult({ "nInserted" : 1 })
> db.blog.insert({"name" : "xwp"});
    WriteResult({ "nInserted" : 1 })
> db.blog.insert({"name" : "xwp"});
    WriteResult({ "nInserted" : 1 })

> db.blog.find()
    { "_id" : ObjectId("550247c18976c0e0b467e800"), "title" : "mytest", "name" : "pc" }
    { "_id" : ObjectId("550248468976c0e0b467e801"), "name" : "xwp" }
    { "_id" : ObjectId("5503ecd83853a67e204962ae"), "title1" : "instrduce of mongo" }
    { "_id" : ObjectId("5503ed873853a67e204962af"), "docment" : "hello mongoDB!" }
    { "_id" : ObjectId("5503edee3853a67e204962b0"), "title" : "mytest", "name" : "pc" }
    { "_id" : ObjectId("5503ee013853a67e204962b1"), "name" : "xwp" }
> db.blog.remove({_id :  ObjectId("550247c18976c0e0b467e800")});
从当前数据库中的blog集合中移除_id为ObjectId("550247c18976c0e0b467e800")的文档
    WriteResult({ "nRemoved" : 1 })
> db.blog.find()
    { "_id" : ObjectId("550248468976c0e0b467e801"), "name" : "xwp" }
    { "_id" : ObjectId("5503ecd83853a67e204962ae"), "title1" : "instrduce of mongo" }
    { "_id" : ObjectId("5503ed873853a67e204962af"), "docment" : "hello mongoDB!" }
    { "_id" : ObjectId("5503edee3853a67e204962b0"), "title" : "mytest", "name" : "pc" }
    { "_id" : ObjectId("5503ee013853a67e204962b1"), "name" : "xwp" }


> show collections;
    blog
    foobar
    system.indexes
> db.createCollection("xunlei");
在当前数据库创建一个指定名称的集合
    { "ok" : 1 }
> show collections;
    blog
    foobar
    system.indexes
    xunlei
>db.blog.findOne()
这个shell函数findOne会返回一个文档 而find函数会返回最多二十个文档.更多区别我们在后面详细介绍.
    { "_id" : ObjectId("550248468976c0e0b467e801"), "name" : "xwp" }
```

2.6. MongoDB的C++/Python驱动安装

2.6.1. 安装pymongo mongoDB的python驱动

```
官网指南
```

```
http://api.mongodb.org/python/current/installation.html

//方法1  安装的最新版本的python驱动(可能不稳定)
$ sudo apt-get install python-dev
$ git clone git://github.com/mongodb/mongo-python-driver.git pymongo
$ cd pymongo/
$ python setup.py install

//方法2(安装自己从官网下载的特定的版本 )https://github.com/mongodb/mongo-python-driver/tags
$ sudo apt-get install build-essential python-dev
$ tar zxvf mongo-python-driver-3.0.tar.gz
$ cd mongo-python-driver-3.0
$ python setup.py install

//方法3
$sudo apt-get install build-essential python-dev pip
$sudo pip install pymongo==3.0
$pip install --upgrade pymongo   更新驱动
```

2.6.2. 下载26compat分支的C++驱动

```
关于C++驱动的介绍:
https://www.mongodb.com/blog/post/introducing-new-c-driver?_ga=1.102106824.890805674.1426172637
MongoDB的 C++驱动目前有三个版本:
    1.26compat .a largely unchanged version of the driver as of MongoDB 2.6 (April 2014) with some
                changes backported from the server.
                下载网址 https://github.com/mongodb/mongo-cxx-driver/tree/26compat
    2.legacy - an improved derivative of the client from the server codebase with a mostly backwards
                compatible interface. This branch will be actively developed and improved through the
                release of  MongoDB 3.2.
                下载网址 https://github.com/mongodb/mongo-cxx-driver/tree/legacy
    3.master - The entirely new and modern MongoDB driver we are introducing in this article.
                下载网址 https://github.com/mongodb/mongo-cxx-driver/tree/master

    版本1,2已经是稳定版本可以根据自己的需求选用,版本3目前还不稳定,不建议使用.
```

```
下载网址:
     https://github.com/mongodb/mongo-cxx-driver/tree/26compat
#git clone https://github.com/mongodb/mongo-cxx-driver.git
#git checkout 26compat
    分支 26compat 设置为跟踪来自 origin 的远程分支 26compat。
    切换到一个新分支 '26compat'
    mongoDB的C++API document
    http://api.mongodb.org/cxx/?_ga=1.77079388.890805674.1426172637
```

```
安装C++驱动之前需要的环境:
    #sudo apt-get update
    #sudo apt-get install scons
    #sudo apt-get install build-essential g++ python-dev autotools-dev libicu-dev libbz2-dev
    #wget –O boost_1_55_0.tar.gz http://sourceforge.net/projects/boost/files/\
    boost/1.55.0/boost_1_55_0.tar.gz
    #tar xzvf boost_1_55_0.tar.gz
    #cd boost_1_55_0/
    #./bootstrap.sh --prefix=/usr/local
    #n=`cat /proc/cpuinfo | grep "cpu cores" | uniq | awk '{print $NF}'`
    #sudo ./b2 --with=all -j $n install
    #sudo ldconfig
    #sudo scons --prefix=$HOME/mongo-client --sharedclient --full install-mongoclient --use-system-boost
```

2.6.3. 下载和安装legacy1.0.0版驱动

```
https://github.com/mongodb/mongo-cxx-driver/tree/legacy
legacy 分支客户端使用例子:
https://github.com/mongodb/mongo-cxx-driver/tree/legacy/src/mongo/client/examples
编译驱动:
    # sudo scons  --prefix=$HOME/mongo-client-legacy --sharedclient install
    # sudo yum install boost*
```

```
legacy C++ driver安装必备:

Boost (>= 1.49) # May work with older versions back to 1.41
Python (2.x)
SCons
Git
# sudo yum install scons
#  wget http://jaist.dl.sourceforge.net/project/boost/boost/1.57.0/boost_1_57_0.zip
# unzip boost_1_17_0.zip
# 查看已经安装boost版本
# rpm -qa|grep boost //卸载掉低于1.49的版本的boost库
# ./bootstrap.sh --prefix=/usr/local
# sudo ./b2 --with=all install

# wget -O mongo-cxx-driver.tar.gz \
https://codeload.github.com/mongodb/mongo-cxx-driver/tar.gz/legacy-1.0.0
# tar zxvf mongo-cxx-driver.tar.gz
//CentOS7通过
# sudo scons --64 --prefix=$HOME/mongo-client-install  --sharedclient install
```

2.6.4. 测试Python驱动

```
#vim mongo_testclient.py
    1 #!/usr/bin/python
    2
    3 import pymongo
    4 import random
    5
    6 conn = pymongo.Connection("127.0.0.1",27017);
    7 db = conn.tutorial
    8 print u'all collection:',db.collection_names()
#ls
        Makefile  mongoclient  mongo_testclient.cpp  mongo_testclient.py
#python  mongo_testclient.py
        all collection: [u'persons', u'system.indexes']
        db.collection_name.find()
        db.collection_name.find({"education":"M.C.A."})
        db.collection.insert(document)
```

2.6.5. 测试C++驱动

```
#sudo vim /etc/profile
    export PATH=$PATH:/home/pc/mongo-client-install/include
    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/pc/mongo-client-install/lib
    source /etc/profile
#vim mongo_testclient.cpp
    28 int main()
    29 {
    30     client::initialize();
    31     DBClientConnection conn;
    32     BSONObj p = BSONObjBuilder().append("name","driver").append("age",33).obj();
    33     try
    34     {
    35         conn.connect("localhost");
    36         cout<<"connect OK"<<endl;
    37     }
    38     catch(DBException &e)
    39     {
    40         cout<<"caught "<<e.what()<<endl;
    41
    42     }
    43     conn.insert("tutorial.persons",p);
    44     conn.insert("tutorial.persons",p);
    45     return 0;
    46 }
```

```
#vim Makefile
    1 CFLAGS = -I /home/pc/mongo-client-install/include/
    2 LDFLAGS = -L/home/pc/mongo-client-install/lib/
    3 all:
    4     g++ -o mongoclient mongo_testclient.cpp $(CFLAGS) $(LDFLAGS)
    5  -lmongoclient -lboost_thread -lboost_filesystem -lboost_program_options
    6  -lboost_system -lboost_regex
#make
g++ -o mongoclient mongo_testclient.cpp $(CFLAGS) $(LDFLAGS) -lmongoclient -lboost_thread \
-lboost_filesystem -lboost_program_options -lboost_system -lboost_regex
#ls
    mongoclient  Makefile  Mongo_testclient.cpp
#./ mongoclient
    connected ok
#mongo
    > use tutorial
    switched to db tutorial
    > show collections;
    persons
    system.indexes
    > db.tutorial.find()
    > db.tutorial.find();
    > db.persons.find()
    { "_id" : ObjectId("5507c9b342eb5bd4292dce1f"), "name" : "driver", "age" : 33 }
    { "_id" : ObjectId("5507c9b342eb5bd4292dce20"), "name" : "driver", "age" : 33 }
```

2.7. 似曾相识的MongoDB

2.7.1. MongoDB VS SQL

The following table presents the various SQL terminology and concepts and the corresponding MongoDB terminology and concepts.

SQL Terms/Concepts	MongoDB Terms/Concepts
database	*database*
table	*collection*
row	*document* or *BSON* document

column	*field*
index	*index*
table joins	embedded documents and linking
primary key Specify any unique column or column combination as primary key.	*primary key* In MongoDB, the primary key is automatically set to the *_id* field.
aggregation (e.g. group by)	aggregation pipeline

Executables¶

The following table presents some database executables and the corresponding MongoDB executables. This table is *not* meant to be exhaustive.

	MongoDB	MySQL	Oracle	Informix	DB2
Database Server	`mongod`	`mysqld`	`oracle`	`IDS`	`DB2 Server`
Database Client	`mongo`	`mysql`	`sqlplus`	`DB-Access`	`DB2 Client`

Examples¶

The following table presents the various SQL statements and the corresponding MongoDB statements. The examples in the table assume the following conditions:

- The SQL examples assume a table named `users`.

- The MongoDB examples assume a collection named `users` that contain documents of the following prototype:

```
{
  _id: ObjectId("509a8fb2f3f4948bd2f983a0"),
  user_id: "abc123",
  age: 55,
  status: 'A'
}
```

Create and Alter¶

The following table presents the various SQL statements related to table-level actions and the corresponding MongoDB statements.

SQL Schema Statements	MongoDB Schema Statements

`CREATE TABLE users ( id MEDIUMINT NOT NULL AUTO_INCREMENT, user_id Varchar(30),age Number,status char(1),PRIMARY KEY (id) )`	Implicitly created on first `insert()` operation. The primary key `_id` is automatically added if `_id` field is not specified.
	`db.users.insert( { user_id: "abc123", age: 55, status: "A" } )`
	However, you can also explicitly create a collection:
	`db.createCollection("users")`

`ALTER TABLE users ADD join_date DATETIME`	Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.

	However, at the document level, `update()` operations can add fields to existing documents using the `$set` operator. `db.users.update( { }, { $set: { join_date: new Date() } }, { multi: true } )`
`ALTER TABLE users DROP COLUMN join_date`	Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level. However, at the document level, `update()` operations can remove fields from documents using the `$unset` operator. `db.users.update( { }, { $unset: { join_date: "" } }, { multi: true } )`
`CREATE INDEX idx_user_id_asc ON users(user_id)`	`db.users.createIndex( { user_id: 1 } )`
`CREATE INDEXidx_user_id_asc_age_desc ON users(user_id, age DESC)`	`db.users.createIndex( { user_id: 1, age: -1 } )`
`DROP TABLE users`	`db.users.drop()`

Insert¶

The following table presents the various SQL statements related to inserting records into tables and the corresponding MongoDB statements.

SQL INSERT Statements	MongoDB insert() Statements

`INSERT INTO users(user_id,`
`age,`
`status)`
`VALUES ("bcd001",45,"A")`
	`db.users.insert(`
	`  { user_id: "bcd001", age: 45, status: "A" }`
	`)`

Select¶

The following table presents the various SQL statements related to reading records from tables and the corresponding MongoDB statements.

SQL SELECT Statements	MongoDB find() Statements
`SELECT *` `FROM users`	`db.users.find()`

`SELECT id,`
`  user_id,`
`  status`
`FROM users`
	`db.users.find(`
	`  { },`
	`  { user_id: 1, status: 1 }`
	`)`

`SELECT user_id, status`
`FROM users`
	`db.users.find(`
	`  { },`
	`  { user_id: 1, status: 1, _id: 0 }`
	`)`

`SELECT *` `FROM users`	`db.users.find(` `  { status: "A" }`

WHERE status = "A"	)
SELECT user_id, status FROM users WHERE status = "A"	db.users.find({ status: "A" }, { user_id: 1, status: 1, _id: 0 })
SELECT * FROM users WHERE status != "A"	db.users.find({ status: { $ne: "A" } })
SELECT * FROM users WHERE status = "A" AND age = 50	db.users.find({ status: "A", age: 50 })
SELECT * FROM users WHERE status = "A" OR age = 50	db.users.find({ $or: [{ status: "A" } , { age: 50 }] })
SELECT * FROM users WHERE age > 25	db.users.find({ age: { $gt: 25 } })
SELECT * FROM users WHERE age < 25	db.users.find({ age: { $lt: 25 } })
SELECT * FROM users WHERE age > 25 AND age <= 50	db.users.find({ age: { $gt: 25, $lte: 50 } })

`SELECT * FROM users WHERE user_id like "%bc%"`	`db.users.find( { user_id: /bc/ } )`
`SELECT * FROM users WHERE user_id like "bc%"`	`db.users.find( { user_id: /^bc/ } )`
`SELECT * FROM users WHERE status = "A" ORDER BY user_id ASC`	`db.users.find( { status: "A" } ).sort( { user_id: 1 } )`
`SELECT * FROM users WHERE status = "A" ORDER BY user_id DESC`	`db.users.find( { status: "A" } ).sort( { user_id: -1 } )`

`SELECT COUNT(*) FROM users`	`db.users.count()`
	or
	`db.users.find().count()`

`SELECT COUNT(user_id) FROM users`	`db.users.count( { user_id: { $exists: true } } )`
	or
	`db.users.find( { user_id: { $exists: true } } ).count()`

`SELECT COUNT(*) FROM users`	`db.users.count( { age: { $gt: 30 } } )`

`WHERE age > 30`	*or* `db.users.find( { age: { $gt: 30 } } ).count()`
`SELECT DISTINCT(status)` `FROM users`	`db.users.distinct( "status" )`

`SELECT *`
`FROM users`
`LIMIT 1`
	`db.users.findOne()`
	or
	`db.users.find().limit(1)`

`SELECT *`
`FROM users`
`LIMIT 5`
`SKIP 10`
	`db.users.find().limit(5).skip(10)`

`EXPLAIN SELECT *`
`FROM users`
`WHERE status = "A"`
	`db.users.find( { status: "A" } ).explain()`

Update Records¶

The following table presents the various SQL statements related to updating existing records in tables and the corresponding MongoDB statements.

SQL Update Statements	MongoDB update() Statements

`UPDATE users`
`SET status = "C"`
`WHERE age > 25`
	`db.users.update(`
	`{ age: { $gt: 25 } },`
	`{ $set: { status: "C" } },`
	`{ multi: true }`

	)
UPDATE users SET age = age + 3 WHERE status = "A"	db.users.update({ status: "A" } , { $inc: { age: 3 } }, { multi: true })

Delete Records¶

The following table presents the various SQL statements related to deleting records from tables and the corresponding MongoDB statements.

SQL Delete Statements	MongoDB remove() Statements
DELETE FROM users WHERE status = "D"	db.users.remove({ status: "D" })
DELETE FROM users	db.users.remove({})

2.7.2. SQL to MongoShell to C++

Several of the C++ driver methods throw mongo::DBException, so you will want a try/catch statement at some level in your program. Also be sure to call `c.getLastError()` after writes to check the error code.

SQL	`manual:mongo` Shell	C++ Driver

INSERT INTO USERS
VALUES(1, 1)
	db.users.insert({ a: 1, b: 1 })	// GENOID is optional. if not done by client,
		// server will add an _id
		
		c.insert("mydb.users",
		BSON(GENOID<<"a"<<1<<"b"<<1));
		// then:
		string err = c.getLastError();

`SELECT a,b FROM users`	`db.users.find( {}, {a: 1, b: 1 } )`	`BSONObj b = BSON("a"<<1<<"b"<<1)`
		`auto_ptr<DBClientCursor> cursor =`
		`  c.query("mydb.users", Query(),`
		`  0, 0, &b);`

`SELECT * FROM users`	`db.users.find()`	`auto_ptr<DBClientCursor> cursor =` `  c.query("mydb.users", Query());`

`SELECT *`
`FROM users`
`WHERE age=33`
	`db.users.find( { age: 33 } )`	`auto_ptr<DBClientCursor> cursor =`
		`  c.query("mydb.users", QUERY("age"<<33))`
		`// or:`
		`auto_ptr<DBClientCursor> cursor =`
		`  c.query("mydb.users", BSON("age"<<33))`

`SELECT *`
`FROM users`
`WHERE age=33`
`ORDER BY name`
	`db.users.find(`
	`{ age: 33 } ).sort( { name: 1 } )`
		`auto_ptr<DBClientCursor> cursor =`
		`  c.query("mydb.users",`
		`  QUERY("age"<<33).sort("name"));`

`SELECT *`
`FROM users`
`WHERE age>33`
`AND age<=40`
	`db.users.find(`
	`{ 'age': { $gt:33, $lte:40 } } )`
		`auto_ptr<DBClientCursor> cursor =`
		`  c.query("mydb.users",`
		`  QUERY("age"<<GT<<33<<LTE<<40));`

`CREATE INDEX myindexname` `ON users(name)`	`db.users.ensureIndex( {name: 1 } )`	`c.ensureIndex("mydb.users", BSON("name"<<1));`

`SELECT *`
`FROM users`
`LIMIT 10`
`SKIP 20`
	`db.users.find().`
	`limit(10).skip(20)`
		`auto_ptr<DBClientCursor> cursor =`
		`  c.query("mydb.users", Query(),`
		`  10, 20);`

`SELECT * FROM users LIMIT 1`	`db.users.findOne()`	`bo obj = c.findOne("mydb.users", Query());`

`SELECT DISTINCT last_name`
`FROM users`
`WHERE x=1`
	`db.users.distinct( 'last_name',`
	`{x: 1} )`
		`// no helper for distinct yet in c++ driver,`
		`// so send command manually`
		`bo cmdResult;`
		`bool ok = c.runCommand(`
		`"mydb",`
		`BSON("distinct" << "users"`
		`<< "key" << "last_name"`
		`<< "query" << BSON("x"<<1)),`
		`cmdResult);`
		`list<bo> results;`
		`cmdResult["values"].Obj().Vals(results);`

`SELECT COUNT(*)`
`FROM users`
`where AGE > 30`
	`db.users.find(`
	`{ age: { $gt: 30 }`
	`} ).count()`
		`unsigned long long n =`
		`c.count("mydb.users", BSON("age"<<GT<<30));`

`UPDATE users`
`SET a=a+2`
`WHERE b='q'`
	`db.users.update( { b: 'q' },`
	`{ $inc: { a:2 } },`
	`false, true)`
		`c.update("mydb.users"`
		`, QUERY("b"<<"q")`
		`,BSON("$inc"<<BSON("a"<<2)), false, true);`
		`// then optionally:`
		`string err = c.getLastError();`
		`bool ok = err.empty();`

`DELETE`
`FROM users`
`WHERE z="abc"`
	`db.users.remove( { z: 'abc' } )`	`c.remove("mydb.users", QUERY("z"<<"abc"));`
		`// then optionally:`
		`string err = c.getLastError();`

See also

Several additional examples, in shell syntax, on the SQL to Mongo Mapping Chart page.

2.7.3. MongoDB C++驱动中的主要类

MongoDB官方提供了对主流语言的驱动程序：

```
    C/C++,Java,C#,Haskell,node.js,PHP,Perl,Python,Ruby等
社区中也提供了其他语言的如Erlang和Go的驱动程序.
legacy分支更多其他类的用法:
    https://github.com/mongodb/mongo-cxx-driver/tree/legacy/src/mongo/client/examples
```

2.7.3.1. BSON相关类

```
mongo::BSONObj: 一个BSON对象(a BSON object) 由多对键值对组成的对象
mongo::BSONElement:在BSON对象中的一个单一成员(a single element in a BSON object.This is a key-value)
mongo::BSONObjBuilder:  用以构造一个BSON对象(used to make BSON objects)
mongo::BSONObjIterator: 用以列举BSON对象(used to enumerate BSON objects)
                        已经被重命名为mongo::iterator //typedef BSONObjIterator iterator;
```

```
BSONtype
    the complete list of valid BSON types
MinKey
    smaller than all other types
EOO
    end of object
NumberDouble
    double precision floating point value
String
    character string, stored in utf8
Object
    an embedded object
Array
    an embedded array
BinData
    binary data
Undefined
    Undefined type.
jstOID
    ObjectId.
Bool
    boolean type
Date
    date type
jstNULL
    null type
RegEx
    regular expression, a pattern with options
DBRef
    deprecated / will be redesigned
Code
```

```
    deprecated / use CodeWScope
Symbol
    a programming language (e.g., Python) symbol
CodeWScope
    javascript code that can execute on the database server, with SavedContext
NumberInt
    32 bit signed integer
Timestamp
    Updated to a Date with value next OpTime on insert.
NumberLong
    64 bit integer
JSTypeMax
    max type that is not MaxKey
MaxKey
    larger than all other types
```

```
BSONElement:
    //作为BSONObj对象的一个成员
    BSONElement (const char *d, int maxLen);
    BSONElement (const char *d);
    BSONElement (const char *d, int fieldNameSize, FieldNameSizeTag)
    BSONElement represents an "element" in a BSONObj.So for the object { a : 3, b : "abc" },
    'a : 3' is the first element (key+value).
    BSONElement代表一个BSON对象的一个成员,对一个一个BSON{a:3,b:"abc"}来说,<a,3>就是一个成员 键值对
    BSONElement     operator[] (const std::string &field) const;
            //retrieve a field within this element throws exception if *this is not an embedded object
    const char *    fieldName () const;  //     field name of the element
    BSONObj Obj();//只作为临时使用
    BSONType type();//返回成员类型
    bool eoo();//判断结尾标志

    BSONObj wrap();//Wrap this element up as a singleton object.
    BSONONj wrap(const StringData &newdata);//以一个新的名字
```

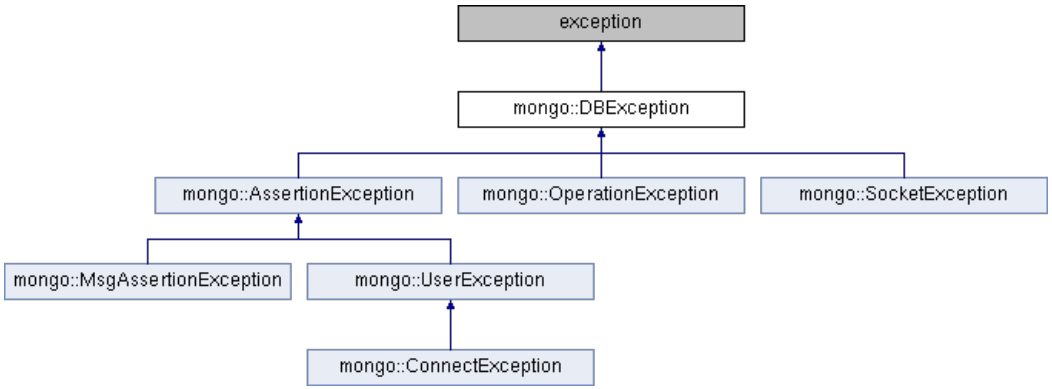
```
BSONObj
    BSON object format://BSON对象的格式
    code <unsigned totalsize=""> {<byte bsontype=""><cstring fieldname=""><Data>}* EOO
        <总大小>{类型key-value}*EOO
        末尾总是带有EOO结束标志,总大小包括自己占用的字节数.totalSize includes itself.
    BSONObj(const char *msgdata);
    BSONOBJ copy() ;//生成一份拷贝
    BSONObj getOwned () const;// isOwned() == true 返回本身 否则调用copy
    bool    isOwned () const ; //判断buffer是否在自己的控制之下
    string  toString( bool isArray, bool full ); //显示出对应jsonstring
```

```
    BSONElement getField(const StringData& name);//根据key获取对应的成员(一个)
    bool    hasField(const StringData& name);//是否有对应的一个key
    bool hasElement(const StringData& name); //同上
    BSONObjIterator begin() const;           //返回BSONObj的开始迭代器

    const char * getStringField(const StringData& name);//获取给定key对应的值 == string 否则输出空
    BSONObj getObjectField(const StringData& name);//return subobject of the given name
    //T       get*T*Field(const StringData& name);//获取指定key对应的值 根据T的不同选择不同类型的API
```

```
BSONObjIterator:

    BSONObjIterator (const BSONObj &jso) //Create an iterator for a BSON object.
    BSONObjIterator (const char *start, const char *end)
    void    operator++ ();
    void    operator++ (int);
    BSONElement     operator* ();
      BSONObjIterator(const BSONObj& o) //Create an iterator for a BSON object.
    bool more() { return _pos < _theend; }  //判断_pos是否已经到达对象末尾
    bool moreWithEOO() { return _pos <= _theend; } //判断_pos是否已经越过对象末尾
    BSONElement next()      //返回当前位置上的成员  然后pos再后移
    BSONElement     next (bool checkEnd)
            //the next element in the object. For the final element, element.eoo() will be true.

    简单用法一般都用于以下场景:
        for(BSONObj::iterator i(o);i.more;)
        {
            //cout<<i.next.toString();
        }
```

```
mongo::BSONObjBuilder
BSONObjBuilder& appendElements(BSONObj x);//添加x对象中所有的元素到当前对象中
BSONObjBuilder& appendElementsUnique( BSONObj x );// 同上 && 要求元素不能存在

BSONObjBuilder& append( const BSONElement& e);//添加一个元素到当前对象中
BSONObjBuilder& appendAs(const BSONElement& e,const StringData& fieldName); //同上&& 以一个新的名字
BSONObjBuilder& append(const StringData& fieldName,BSONObj subObj);//添加一个子对象作为成员
BSONObjBuilder& append(const StringData& fieldName,T &value);      //

BSONObjBuilder& appendTimetamp(const String& fieldName);//添加一个时间戳
BSONObj obj();
BSONObjBuilder& operator<<(const BSONElement &e);
BSONObjBuilder& genOID(){return append("_id",OID::gen)};   //生成一个_id
```

```
驱动连接远程主机进行操作
1.更改远程主机的配置文件
sudo vim /etc/mongod.conf
#bind_ip = 127.0.0.1        //在bind_ip = 127.0.0.1前面加上'#'
2.需要验证写操作是否成功,应该在每次写操作之后
string s = dbconnect.getLastError();
如果s为空 s.empty() == true就代表写操作成功
    如果需要获取详细错误,就调用  getLastErrorDetailed()
3.捕获异常
DBException相关类继承结构解析
```



virtual const char * what ()

const throw () try{}catch(DBException &e) { cout<<e.what()<<endl; }

```
BSON使用的示例:
    /* make a bit more complex object with some nesting
       { x : 'asdf', y : true, subobj : { z : 3, q : 4 } }
    */
    bo y = BSON( "x" << "asdf" << "y" << true << "subobj" << BSON( "z" << 3 << "q" << 4 ) );

    /* print it */
    cout << "y: " << y << endl;

    /* reach in and get subobj.z */
    cout << "subobj.z: " << y.getFieldDotted("subobj.z").Number() << endl;

    /* alternate syntax: */
    cout << "subobj.z: " << y["subobj"]["z"].Number() << endl;

    BSONObj p = BSONObjBuilder().append("name", "Joe").append("age", 33).obj();
    BSONObj p = BSON( "name" << "Joe" << "age" << 33 );
```

2.7.3.2. DBclientConnection类(连接到MongoDB上进行操作的类)

```
对这个连接进行操作就相当于对数据库进行的操作.
DBClientConnection (bool _autoReconnect=false, DBClientReplicaSet *cp=0, double so_timeout=0)
virtual bool    connect (const HostAndPort &server, std::string &errmsg)
        Connect to a Mongo database server.
bool    connect (const std::string &server, std::string &errmsg)
    Compatibility connect now that HostAndPort has an explicit constructor.
void    connect (const std::string &serverHostname)
    Connect to a Mongo database server.
virtual void    logout (const std::string &dbname, BSONObj &info)
    Logs out the connection for the given database.
virtual std::auto_ptr< DBClientCursor >     query (const std::string &ns, Query query=Query(),
                                int nToReturn=0, int nToSkip=0, const BSONObj *fieldsToReturn=0,
                                int queryOptions=0, int batchSize=0)
    send a query to the database.
        ns  namespace to query, format is <dbname>.<collectname>[.<collectname>]*
        query           query to perform on the collection. this is a BSONObj (binary JSON)
                        You may format as { query: { ... }, orderby: { ... } } to specify a sort order.
        nToReturn       n to return (i.e., limit). 0 = unlimited
        nToSkip         start with the nth item
        fieldsToReturn  optional template of which fields to select. if unspecified, returns all fields
        queryOptions    see options enum at top of this file
        Returns         cursor. 0 if error (connection failure)

virtual unsigned long long  query (stdx::function< void(DBClientCursorBatchIterator &)> f,
            const std::string &ns, Query query, const BSONObj *fieldsToReturn, int queryOptions)

virtual bool    runCommand (const std::string &dbname, const BSONObj &cmd, BSONObj &info, int options=0)
    Run a database command
std::string     toString () const

std::string     getServerAddress () const

const HostAndPort &     getServerHostAndPort () const

virtual void    killCursor (long long cursorID);//显式的杀死游标

virtual unsigned long long  count (const std::string &ns, const Query &query=Query(), int options=0,
    int limit=0, int skip=0);//获取文档的个数

virtual void    insert (const std::string &ns, BSONObj obj, int flags=0, const WriteConcern *wc=NULL)
                insert an object into the database
virtual void    insert (const std::string &ns, const std::vector< BSONObj > &v, int flags=0,
    const WriteConcern *wc=NULL)
            insert a vector of objects into the database
```

```
virtual void    update (const std::string &ns, Query query, BSONObj obj, bool upsert=false,
    bool multi=false, const WriteConcern *wc=NULL)
            updates objects matching query

virtual void    update (const std::string &ns, Query query, BSONObj obj, int flags,
    const WriteConcern *wc=NULL)
virtual void    remove (const std::string &ns, Query q, bool justOne=0, const WriteConcern *wc=NULL)
            remove matching objects from the database More...

virtual void    remove (const std::string &ns, Query query, int flags, const WriteConcern *wc=NULL)


void createIndex(   const StringData &  ns,const BSONObj &  keys )  inline;
    Create an index on the collection 'ns' as described by the given keys.
    If you wish to specify options, see the more flexible overload of 'createIndex' which takes an
    IndexSpec object. Failure to construct the index is reported by throwing an OperationException.
    Parameters
    ns  Namespace on which to create the index
    keys    Document describing keys and index types.
            You must provide at least one field and its direction
virtual void    dropIndex (const std::string &ns, BSONObj keys);

virtual bool dropCollection(const std::string & ns,BSONObj * info= NULL);
std::string     getLastError (const std::string &db, bool fsync=false, bool j=false, int w=0,
    int wtimeout=0)
virtual BSONObj     getLastErrorDetailed (const std::string &db, bool fsync=false, bool j=false,
    int w=0, int wtimeout=0)
    Get error result from the last write operation (insert/update/delete) on this connection.
```

2.7.3.3. DBClientCursor类(游标)

```
    DBClientCursor (DBClientBase *client, const std::string &_ns, BSONObj _query, int _nToReturn,
                        int _nToSkip, const BSONObj *_fieldsToReturn, int queryOptions, int bs)
    DBClientCursor (DBClientBase *client, const std::string &_ns, long long _cursorId,
                        int _nToReturn,int options, int _batchSize)
    bool    more ()
        If true, safe to call next(). More...
    BSONObj     next ()
        next More...
    void    putBack (const BSONObj &o)
        restore an object previously returned by next() to the cursor
    BSONObj     nextSafe ()
        throws AssertionException if get back { $err : ...

    void    peek (std::vector< BSONObj > &, int atMost)
```

```
        peek ahead at items buffered for future next() calls.
    BSONObj     peekFirst ()
        挑出第一个成员
    bool    peekError (BSONObj *error=NULL)
        peek ahead and see if an error occurred, and get the error if so.

    int     itcount ()
        iterate the rest of the cursor and return the number if items

    bool    isDead () const
        cursor no longer valid – use with tailable cursors.
    std::string     getns () const
    long long       getCursorId () const
    void    decouple();//by default we "own" the cursor and will send the server a KillCursor message
                    when ~DBClientCursor() is called.
```

2.7.3.4. Query相关类

```
Typically one uses the MONGO_QUERY(...) macro to construct a Query object.
Examples:
            MONGO_QUERY( "age" << 33 << "school" << "UCLA" ).sort("name")
            MONGO_QUERY( "age" << GT << 30 << LT << 50 )

    Query (const BSONObj &b)
    Query (const std::string &json)
    Query (const char *json)

    Query &     sort (const BSONObj &sortPattern)
    Query &     sort (const std::string &field, int asc=1)//
        Add a sort (ORDER BY) criteria to the query expression. More...
    Query &     hint (BSONObj keyPattern)
    Query &     hint (const std::string &indexName)
        Provide a hint to the query.
    Query &     maxTimeMs (int millis)
        Specifies a cumulative time limit in milliseconds for processing an operation.
    Query &     minKey (const BSONObj &val)
        Provide min and/or max index limits for the query. More...
    Query &     maxKey (const BSONObj &val)
        max is exclusive

    Query &     explain ()
        Return explain information about execution of this query instead of the actual query results.

    Query &     snapshot ()
        Use snapshot mode for the query.
    Query &     where (const std::string &jscode, BSONObj scope)
```

```
        Queries to the Mongo database support a $where parameter option which contains  a javascript
        function that is evaluated to see whether objects being queried match its criteria.
    Query &     where (const std::string &jscode)

    bool    isComplex (bool *hasDollar=0) const

    BSONObj     getModifiers () const

    BSONObj     getFilter () const

    BSONObj     getSort () const

    BSONElement     getHint () const

    BSONObj     getReadPref () const

    int     getMaxTimeMs () const

    bool    isExplain () const

    bool    hasReadPreference () const
    bool    hasHint () const
    bool    hasMaxTimeMs () const

    std::string     toString () const
    operator std::string () const
```

2.7.4. mongoDB的python驱动相关方法

```
>>>import datetime #导入日期相关
>>>import pymongo  #导入pymongo(在python驱动章节已经安装) && 远程或者本地已经安装好mongoDB
>>>from pymongo import MongoClient
#使用MongoClient连接到 mongod示例
>>> client = MongoClient() #连接本地数据库
>>> client = MongoClient('192.168.22.136', 27017)  #连接远程数据库
>>> client = MongoClient('mongodb://localhost:27017/')
使用方法:

class pymongo.mongo_client.MongoClient(host='localhost', port=27017, document_class=dict,
                                       tz_aware=False, connect=True, **kwargs)
http://api.mongodb.org/python/current/api/pymongo/mongo_client.html#pymongo.mongo_client.MongoClient
Parameters:
host (optional): hostname or IP address of the instance to connect to, or a mongodb URI,
            or a list of hostnames / mongodb URIs.
```

```
port (optional): port number on which to connect
document_class (optional): default class to use for documents returned from queries on this client
tz_aware (optional): if True, datetime instances returned as values in a document by this MongoClient
            will be timezone aware (otherwise they will be naive)
connect (optional): if True (the default), immediately begin connecting to MongoDB in the background.
            Otherwise connect on the first operation.
Other optional parameters can be passed as keyword arguments:
maxPoolSize (optional): The maximum number of connections that the pool will open simultaneously.
    If  this is set, operations will block if there are maxPoolSize outstanding connections from the
    pool.   Defaults to 100.
socketTimeoutMS: (integer or None) How long (in milliseconds) a send or receive on a socket can take
    before timing out. Defaults to None (no timeout).
connectTimeoutMS: (integer or None) How long (in milliseconds) a connection can take to be opened
    before  timing out. Defaults to 20000.
MongoClient的属性:
    close()
        Disconnect from MongoDB.//断开与mongodb的连接
        Close all sockets in the connection pools and stop the monitor threads.
        //断掉所有连接 停止monotor线程
        If this instance is used again it will be automatically re-opened and the threads restarted.

    c[db_name] || c.db_name
        Get the db_name Database on MongoClient c.
    address
        (host, port) of the current standalone, primary, or mongos, or None.
    max_pool_size
        The maximum number of sockets the pool will open concurrently.超过最这个限制将阻塞新的操作
    database_names()
        Get a list of the names of all databases on the connected server.
    get_default_database()
        Get the database named in the MongoDB connection URI.//获取使用URI格式连接的数据库名字
        >>> uri = 'mongodb://host/my_database'
        >>> client = MongoClient(uri)
        >>> db = client.get_default_database()
        >>> assert db.name == 'my_database'
    get_database(name, codec_options=None, read_preference=None, write_concern=None)
        Get a Database with the given name and options.

    server_info()
        Get information about the MongoDB server we're connected to.
    close_cursor(cursor_id, address=None)
            Close a single database cursor.
    Parameters:
        cursor_id: id of cursor to close
        address (optional): (host, port) pair of the cursor's server.If it is not provided,
            the client attempts to close the cursor on the primary or standalone, or a mongos server.
```

```
            Changed in version 3.0: Added address parameter.

    kill_cursors(cursor_ids, address=None)
        Send a kill cursors message soon with the given ids.
        调用Cursor的析构函数在垃圾回收阶段,用以 锁或者网络IO不安全.定时会被后台线程关闭.
    Parameters:
        cursor_ids: list of cursor ids to kill
        address (optional): (host, port) pair of the cursor's server. If it is not provided,
            the client attempts to close the cursor on the primary or standalone, or a mongos server.

#获取数据库database
>>> db = client.test_database
>>> db = client['test-database']

#获取集合collection
>>> collection = db.test_collection
>>> collection = db['test-collection']

#文档document
>>> post = {"author": "Mike",
...         "text": "My first blog post!",
...         "tags": ["mongodb", "python", "pymongo"],
...         "date": datetime.datetime.utcnow()}
>>> post_id = collection.insert_one(post).inserted_id
//当一个文档插入时 如果并不包含_id则自动增加. _id对应的value需要在整个集合中唯一.
//insert_one()返回一个InsertOneResult的实例.

db.collection_names(include_system_collections=False)   #获取db所有集合名(是否包含系统集合名)
```

```
#insert(doc_or_docs, manipulate=True, check_keys=True, continue_on_error=False, **kwargs)
    Insert a document(s) into this collection.
    DEPRECATED - Use insert_one() or insert_many() instead.
#update(spec, document, upsert=False, manipulate=False, multi=False, check_keys=True, **kwargs)
    Update a document(s) in this collection.
    DEPRECATED - Use replace_one(), update_one(), or update_many() instead.
#remove(spec_or_id=None, multi=True, **kwargs)
    Remove a document(s) from this collection.
    DEPRECATED - Use delete_one() or delete_many() instead.
#find_and_modify(query={}, update=None, upsert=False, sort=None, full_response=False,
                manipulate=False, **kwargs)
    Update and return an object.
    DEPRECATED - Use find_one_and_delete(), find_one_and_replace(), or find_one_and_update() instead

#ensure_index(key_or_list, cache_for=300, **kwargs)
    DEPRECATED - Ensures that an index exists on this collection.
```

```
#find_one获取单一文档
find_one(filter_or_id=None, *args, **kwargs)

参数:
    filter (optional): a dictionary specifying the query to be performed OR any other type
        to be used as the value for a query for "_id".
    *args (optional): 同find().
    **kwargs (optional): 同find().
    max_time_ms (optional): a value for max_time_ms may be specified as part of **kwargs,
返回值:
    //Get a single document from the database.如果没有匹配的文档 返回空


>>> collection.find_one()
>>> collection.find_one({"author": "Mike"})
>>> posts.find_one({"_id": post_id}) #如果知道某个文档的_id值为post_id


>>> from bson.objectid import ObjectId
    The web framework gets post_id from the URL and passes it as a string
def get(post_id):   #转换字符串post_id为ObjectID类型 Convert from string to ObjectId:
    document = client.db.collection.find_one({'_id': ObjectId(post_id)})
```

```
#find(filter=None, projection=None,....) 用于查找多文档
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)
     第一个参数为匹配文档的条件,必须满足.
    返回值:
        returns a Cursor instance, which allows us to iterate over all matching documents.
    >>> for doc in collection.find():
    ...     doc


Parameters:
filter (optional):a SON object specifying elements which must be present for a document
            to be included in the result set
projection (optional): a list of field names that should be returned in the result set or
    a dict  specifying the fields to include or exclude. If projection is a list “_id” will always
    be returned. Use a dict to exclude fields from the result
            (e.g. projection={‘_id’: False}).
skip (optional): the number of documents to omit (from the start of the result set)
            when returning the results
```

```
limit (optional): the maximum number of results to return

>>> for doc in db.test.find({}):
...     print(doc)
```

```
#count()统计文档的个数
    Get the number of documents in this collection.
    >>> collection.count()
    >>> collection.find({"author": "Mike"}).count()


Parameters:
    filter (optional): A query document that selects which documents to count in the collection.
    **kwargs (optional):

    See list of options.
    hint (string or list of tuples):
        The index to use. Specify either the index name as a string or the index specification as
        a list of tuples (e.g. [('a', pymongo.ASCENDING), ('b', pymongo.ASCENDING)]).
    limit (int): The maximum number of documents to count.
    skip (int): The number of matching documents to skip before returning results.
    maxTimeMS (int): The maximum amount of time to allow the count command to run, in milliseconds.
```

```
#insert_many(documents, ordered=True)  #插入文档的列表(多文档)Insert a list of documents.

Parameters:
    documents: A list of documents to insert.
    ordered (optional): If True (the default) documents will be inserted on the server serially,
        in the order provided. If an error occurs all remaining inserts are aborted.
        If False, documents will be inserted on the server in arbitrary order, possibly in parallel,
        and all document inserts will be attempted.
Returns:
    An instance of InsertManyResult.

    >>> db.test.count()
    0
    >>> result = db.test.
    >insert_many([{'x': i} for i in range(2)])
    >>> result.inserted_ids
    [ObjectId('54f113fffba522406c9cc20e'), ObjectId('54f113fffba522406c9cc20f')]
    >>> db.test.count()
    2
#bulk_write(requests, ordered=True)
    Send a batch of write operations to the server.
```

```
    >>> for doc in db.test.find({}):
    ...     print(doc)
    ...
    {u'x': 1, u'_id': ObjectId('54f62e60fba5226811f634ef')}
    {u'x': 1, u'_id': ObjectId('54f62e60fba5226811f634f0')}
    >>> # DeleteMany, UpdateOne, and UpdateMany are also available.
    ...
    >>> from pymongo import InsertOne, DeleteOne, ReplaceOne
    >>> requests = [InsertOne({'y': 1}), DeleteOne({'x': 1}),
    ...             ReplaceOne({'w': 1}, {'z': 1}, upsert=True)]
    >>> result = db.test.bulk_write(requests)
    >>> result.inserted_count
    1
    >>> result.deleted_count
    1
    >>> result.modified_count
    0
    >>> result.upserted_ids
    {2: ObjectId('54f62ee28891e756a6e1abd5')}
    >>> for doc in db.test.find({}):
    ...     print(doc)
    ...
    {u'x': 1, u'_id': ObjectId('54f62e60fba5226811f634f0')}
    {u'y': 1, u'_id': ObjectId('54f62ee2fba5226811f634f1')}
    {u'z': 1, u'_id': ObjectId('54f62ee28891e756a6e1abd5')}
#update_one(filter, update, upsert=False)
    Update a single document matching the filter.
    Parameters:
        filter: A query that matches the document to update.
        update: The modifications to apply.
        upsert (optional): If True, perform an insert if no documents match the filter.
        Returns:    An instance of UpdateResult.
    >>> for doc in db.test.find():
    ...     print(doc)
    ...
    {u'x': 1, u'_id': 0}
    {u'x': 1, u'_id': 1}
    {u'x': 1, u'_id': 2}
    >>> result = db.test.update_one({'x': 1}, {'$inc': {'x': 3}})
    >>> result.matched_count
    1
    >>> result.modified_count
    1
    >>> for doc in db.test.find():
    ...     print(doc)
    ...
```

```
    {u'x': 4, u'_id': 0}
    {u'x': 1, u'_id': 1}
    {u'x': 1, u'_id': 2}
#delete_one(filter)
    Delete a single document matching the filter.
    Parameters:
        filter:
                A query that matches the document to delete.
    Returns:    An instance of DeleteResult.
    >>> db.test.count({'x': 1})
    3
    >>> result = db.test.delete_one({'x': 1})
    >>> result.deleted_count
    1
    >>> db.test.count({'x': 1})
    2
#delete_many(filter)
    Delete one or more documents matching the filter.

    >>> db.test.count({'x': 1})
    3
    >>> result = db.test.delete_many({'x': 1})
    >>> result.deleted_count
    3
    >>> db.test.count({'x': 1})
    0
    Parameters:
        filter: A query that matches the documents to delete.
    Returns:    An instance of DeleteResult.

#create_index(keys, **kwargs)   #创建索引优化集合的查询效率
        Takes either a single key or a list of (key, direction) pairs.
        The key(s) must be an instance of basestring (str in python 3), and the direction(s) must be
        one of (ASCENDING, DESCENDING, GEO2D, GEOHAYSTACK, GEOSPHERE, HASHED, TEXT).
    >>> my_collection.create_index("mike")
    >>> my_collection.create_index([("mike", pymongo.DESCENDING),("eliot", pymongo.ASCENDING)])
    >>> my_collection.create_index([("mike", pymongo.DESCENDING)],background=True)
#create_indexes(indexes)    #创建多个索引
        Create one or more indexes on this collection.
    >>> from pymongo import IndexModel, ASCENDING, DESCENDING
    >>> index1 = IndexModel([("hello", DESCENDING),
    ...                      ("world", ASCENDING)], name="hello_world")
    >>> index2 = IndexModel([("goodbye", DESCENDING)])
    >>> db.test.create_indexes([index1, index2])
    ["hello_world"]
#drop_index(index_or_name)  #删除索引
```

```
    Drops the specified index on this collection
    Parameters:
        index_or_name: index (or name of index) to drop
#drop_indexes() #删除集合上的所有索引
    Drops all indexes on this collection.
#reindex()
    Rebuilds all indexes on this collection.
#list_indexes()
    Get a cursor over the index documents for this collection.
>>> for index in db.test.list_indexes():
...     print(index)
...
    SON([(u'v', 1), (u'key', SON([(u'_id', 1)])),
    (u'name', u'_id_'), (u'ns', u'test.test')])
#drop()         #删除某个集合
>>> db.foo.drop()
>>> db.drop_collection("foo")
#rename(new_name, **kwargs)  #重命令集合
    Rename this collection.
Parameters:
    new_name:
        new name for this collection
    **kwargs (optional):
        additional arguments to the rename command may be passed as keyword arguments to
        this helper method (i.e. dropTarget=True)
#group(key, condition, initial, reduce, finalize=None, **kwargs) #分组
    Returns an array of grouped items.

parameters:
    key: fields to group by (see above description)
    condition: specification of rows to be considered (as a find() query specification)
    initial: initial value of the aggregation counter object
    reduce: aggregation function as a JavaScript string
    finalize: function to be called on each object in output list.
    **kwargs (optional): additional arguments to the group command may be passed
                         as keyword arguments to this helper method
```

2.8. mongoDB之ObjectID

```
http://docs.mongodb.org/manual/reference/object-id/
"_id"所代表的字符串并不是以字符串形式存在
y = ObjectId("507f191e810c19729de860ea")
ObjectId("507f191e810c19729de860ea")

ObjectId("507f191e810c19729de860ea").getTimestamp()
```

```
ISODate("2012-10-17T20:46:22Z")

ObjectId("507f191e810c19729de860ea").str
507f191e810c19729de860ea

ObjectId("507f191e810c19729de860ea").valueOf()
507f191e810c19729de860ea

ObjectId("507f191e810c19729de860ea").toString()
ObjectId("507f191e810c19729de860ea")
```

2.9. pymongo之results

- pymongo.results.BulkWriteResult(bulk*api*result, acknowledged)

  Parameters: bulk*api*result: A result dict from the bulk API acknowledged: Was this write result acknowledged? If False then all properties of this object will raise InvalidOperation. bulk*apiresult upsertedidsThe raw bulk API result. deleted*count The number of documents deleted. inserted*count The number of documents inserted. matched*count The number of documents matched for an update. modified_count The number of documents modified. upserted*count The number of documents upserted. upserted*ids A map of operation index to the _id of the upserted document.

- pymongo.results.DeleteResult(raw_result, acknowledged)

  The return type for delete*one() and delete*many() deleted*count The number of documents deleted. raw*result The raw result document returned by the server.

- pymongo.results.InsertManyResult(inserted_ids, acknowledged)

  The return type for insert*many() inserted*ids A list of _ids of the inserted documents, in the order provided.

- pymongo.results.InsertOneResult(inserted_id, acknowledged)

  The return type for insert*one(). inserted*id The inserted document's _id

- pymongo.results.UpdateResult(raw_result, acknowledged)

  matched count The number of documents matched for this update.

modified*count The number of documents modified. raw*result The raw result document returned by the server. upserted_id The _id of the inserted document if an upsert took place. Otherwise None. Create a BulkWriteResult instance.

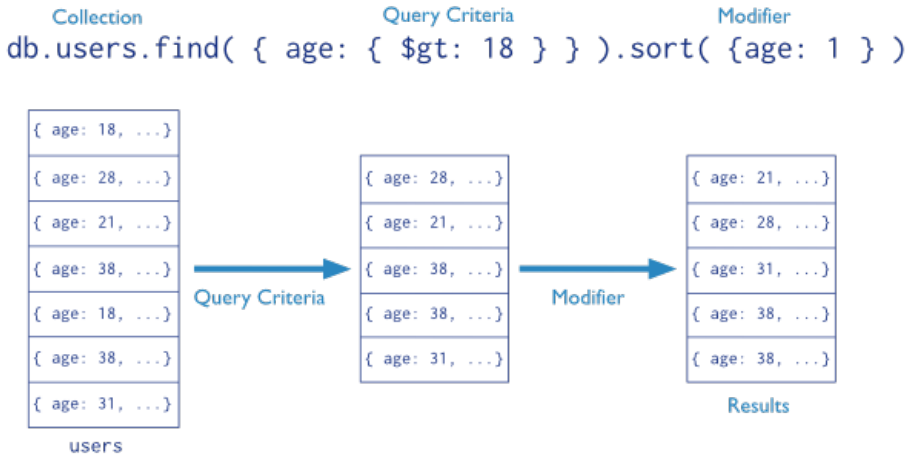
3. MongoDB的增删改查操作(MongoDB CRUD Operations)

```
MongoDB provides rich semantics for reading and manipulating data. CRUD stands for create, read,
update, and delete. These terms are the foundation for all interactions with the database.
mongoDB提供了丰富的语法用以读取和操作数据.
CRUD代表了创建 读取 更新 删除四个操作，这几个用语是所有与数据库的交互的基础.
```

3.1. MongoDB CRUD简介

```
MongoDB存储的数据格式是文档 document,类似于JSON的键值对:
```


```
MongoDB将文档存储在集合 collection中:
```

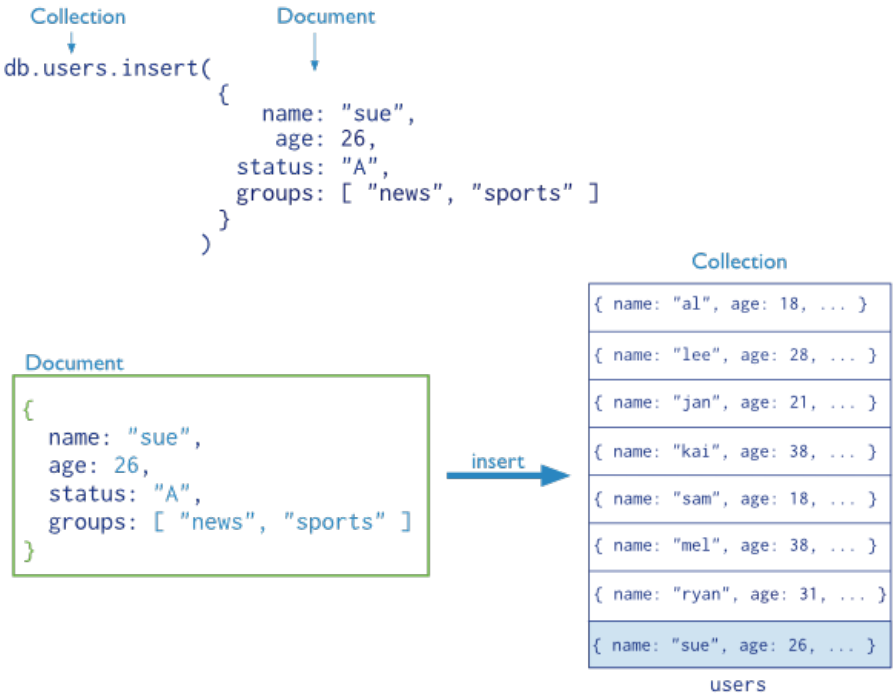


3.2. Query Database Operations 查询操作

```
In MongoDB a query targets a specific collection of documents.Queries specify criteria, or conditions,
that identify the documents that MongoDB returns to the clients. A query may include a projection that
specifies the fields from the matching documents to return. You can optionally modify queries to impose
limits, skips, and sort orders. In the following diagram, the query process specifies a query criteria
and a sort modifier.

在MongoDB中查询目标是特定文档的集合。查询指定条件,确定MongoDB的返回到客户端的文件一个查询可能包括一个指定字段
的来自返回匹配的文档。您可以修改查询限制,跳过,排序.在下图中,查询过程指定查询条件和修饰符.
在MongoDB中查询的方法是:
    查找多文档db.collection.find() ,单文档查找db.collection.findOne()
```

3.3. data modifcation Operations 数据修改操作

```
    修改数据操作指的的是
    db.collection.create() 创建集合
    db.collection.update() 更新集合中给定条件的文档
    db.collection.remove() 移除给定条件的文档
    db.collection.drop()   移除整个集合
    这些操作是针对单一集合的操作.
```



3.4. mongo Shell Methods

```
mongoshell是一种与mongoDB交互的工具,语法与JavaScript相似.mongo Shell Method就是在mongoshell上使用的方法.
分别针对集合,游标,数据库,用户管理,分片,副本集.由于方法太多这里不一一将其列出.只列取其中几个初级使用的方法.
点击网站获取更多:
http://docs.mongodb.org/manual/reference/method/js-collection/

针对集合操作的方法:
db.collection.insert()  Creates a new document in a collection.
db.collection.remove()  Deletes documents from a collection.
db.collection.drop()    Removes the specified collection from the database.
db.collection.update()  Modifies a document in a collection.
db.collection.find()    Performs a query on a collection and returns a cursor object.
db.collection.findAndModify()   Atomically modifies and returns a single document.
db.collection.findOne() Performs a query and returns a single document.

db.collection.aggregate()   Provides access to the aggregation pipeline.
db.collection.count()   return a count of the number of documents in a collection or matching a query.

db.collection.dropIndex()   Removes a specified index on a collection.
db.collection.dropIndexes() Removes all indexes on a collection.
```

3.4.1. 查找多文档db.collection.find(query, projection)

```
Selects documents in a collection and returns a cursor to the selected documents
选择在集合中的文档并且返回一个指向被选择到的文档的游标.

find()函数有以下两个参数
```

- 参数query 使用查询操作符指定查询条件,默认第一个参数{}匹配所有,并且显示20条document
- 参数projection 以一个文档的形式指定返回的域(忽略这个参数将返回所有域) ,文档的形式如下: { field1: boolean, field2: boolean, ... }

  可以通过设置第二个参数的值来改变返回数据的属性field:1/true 为需要 field:0/flase为不需要. 例如：*“id”:0 表示不需要*id

  > db.blog.find({},{"_id":0,"name":1}) { "name" : "pc" } { } { "name" : "pyy" }

  MongoDB中比较操作符有： (>) 大于 $gt
  (<) 小于 $lt
  (>=) 大于等于 $gte (<=) 小于等于 $lte (!=) 不等于 $ne 可匹配任意类型 以上为全部的比较操作运算符g-Great e-Equal l-Little 找出没有几个同学(<60): db.blog.find({score:{$lt:60}},{*"id":0,"name":1}) 找出成绩为优秀的同学(>=90): db.blog.find({score:{$gte:90}},{"*id":0,"name":1})

3.4.2. 查找单文档db.collection.findOne(query, projection)

```
Returns one document that satisfies the specified query criteria. If multiple documents satisfy
the query, this method returns the first document according to the natural order which reflects the
order of documents on the disk. In capped collections, natural order is the same as insertion order.
返回一个文档(满足指定条件),如果多个文档都满足条件,将根据在磁盘上的顺序返回第一个文档.
在固定集合上,磁盘顺序和插入顺序一致.

参数意思和find()函数一致
```

3.4.3. 统计个数db.collection.count(query)

```
参数query表示选择条件,忽略query表示没有条件
    The db.collection.count(query) ==  db.collection.find(query).count()
>db.collection.find( { a: 5, b: 5 } ).count()
```

```
等效于
>db.collection.find( { a: 5, b: 5 } )

统计orders集合中 ord_dt时间比01/01/2012时间晚的文档数量
>db.orders.count( { ord_dt: { $gt: new Date('01/01/2012') } } )
```

3.4.4. 插入新文档db.collection.insert(document..)

```
insert()函数结构:
    db.collection.insert(
       <document or array of documents>,
       {
         writeConcern: <document>,
         ordered: <boolean>
       }
    )
参数:
document    一个需要被插入集合的文档或者文档的数组
writeConcern  (一般翻译作)安全写级别.A document expressing the write concern.
              Omit to use the default write concern. See Safe Writes.
        默认情况下，该操作会使用WriteConcern.NORMAL选项
        WriteConcern.NONE:没有异常抛出
        WriteConcern.NORMAL:仅抛出网络错误异常，没有服务器错误异常
        WriteConcern.SAFE:抛出网络错误异常、服务器错误异常；并等待服务器完成写操作。
        WriteConcern.MAJORITY: 抛出网络错误异常、服务器错误异常；并等待一个主服务器完成写操作。
        WriteConcern.FSYNC_SAFE: 抛出网络错误异常、服务器错误异常；写操作等待服务器将数据刷新到磁盘。
        WriteConcern.JOURNAL_SAFE:抛出网络错误异常、服务器错误异常；写操作等待服务器提交到磁盘的日志文件。
        WriteConcern.REPLICAS_SAFE:抛出网络错误异常、服务器错误异常；等待至少2台服务器完成写操作。

ordered     (默认)真执行一个有序插入,否则为无序的插入(过程中有错误,将继续插入后面的文档)
If true, perform an ordered insert of the documents in the array, and if an error occurs
with one of documents, MongoDB will return without processing the remaining documents in the array.
If false,perform an unordered insert, and if an error occurs with one of documents, continue processing
the remaining documents in the array.   Defaults to true
```

3.4.5. 移除集合db.collection.drop()

```
删除整个集合(直接删除集合中所有的文档) 这个操作会迅速的完成.
>db.blog.drop()
```

3.4.6. 移除文档db.collection.remove()

```
db.collection.remove(
   <query>,
```

```
   <justOne>
)
db.collection.remove(
   <query>,
   {
     justOne: <boolean>,
     writeConcern: <document>
   }
)
移除所有的文档 这个操作会一条一条的删除mongoDB的文档  ***很慢
>db.bios.remove( { } )
删除products集合中 所有qty>20的文档
>db.products.remove( { qty: { $gt: 20 } } )
删除符合条件的第一条
>db.products.remove( { qty: { $gt: 20 } }, true )
```

3.4.7. 更新db.collection.update(criteria, objNew, upsert, multi)

```
update()函数接受以下四个参数：
- criteria : update的查询条件，类似sql update查询内where后面的。

- objNew : update的对象和一些更新的操作符（如$,- ...）等，也可以理解为sql update查询内set后面的

- upsert : 这个参数的意思是，如果不存在update的记录，是否插入objNew,true为插入，默认是false，不插入。

- multi : mongodb默认是false,只更新找到的第一条记录，如果这个参数为true,  就把按条件查出来多条记录全部更新。
>db.blog.update({score:{$lt:60}},{$inc:{score:30}},false,true);
```

3.4.8. 创建索引db.collection.createIndex(keys, options)

```
在mongoDB3.0版本中弃用了ensureIndex(key,options)函数创建索引.
keys    键值对的形式  值对文档中包含的键(作为索引)进行描述,升序为1,降序为-1.
                A document that contains the field and value pairs where the field is the index key
        and the value describes the type of index for that field. For an ascending index on a field,
        specify a value of 1; for descending index, specify a value of -1.

        MongoDB支持一些不同的索引类型,包括全文,地理,哈希.
        MongoDB supports several different index types including text, geospatial, and hashed indexes.
                        See Index Types for more information.

options选项   以一个文档的形式包含选项的集合 用以控制索引的产生.options   Optional. A document that
contains a set of options that controls the creation of the index.
通用索引的常用选项:
background      如果当前机器负荷大推荐以这种选项创建索引.默认为否
unique          如果带有该选项创建索引,mongoDB将不会接受新的文档的索引域的值已经在数据库中存在.
```

```
                该选项对于hash索引无效.
name            指定索引的名称
storageEngine   指定数据存储引擎{ <storage-engine-name>: <options> }  3.0新增
    其他索引的对应的选项请查阅官方文档.
>db.collection.createIndex( { orderDate: 1, zipcode: -1 } )
```

3.4.9. 删除单个索引db.collection.dropIndex(index)

```
            You cannot drop the default index on the _id field.
            Specifies the index to drop. You can specify the index either by the index name or by
            the index specification document. [1]
        删除指定的索引.你可以指定索引名或指定索引的key所对应文档
        使用db.collection.getIndexes() 获取索引的信息(key:xxx,name:xxx)
            ***To drop a text index, specify the index name.
        db.pets.dropIndex( "catIdx" ) 或者 db.pets.dropIndex( { "cat" : -1 } )
```

3.4.10. 删除多个索引db.collection.dropIndexs()

```
        Drops all indexes other than the required index on the _id field.
        Only call dropIndexes() as a method on a collection object.
        删除所有索引除了_id域上的索引.
```

3.4.11. 获取信息db.collection.explain()

```
        New in version 3.0.
        Returns information on the query plan for the following operations:
        mongo3.0版本中新加的方法，返回下列操作查询计划的信息：
        aggregate(); count(); find(); group(); remove(); and update() methods
        db.collection.explain().<method(...)>
        示例：
            db.products.explain().remove( { category: "apparel" }, { justOne: true } )
        参数
        verbosity    The possible modes are: "queryPlanner", "executionStats", and "allPlansExecution".
                     Default mode is "queryPlanner".
        queryPlanner Mode:
        >db.products.explain().count( { quantity: { $gt: 50 } } )

        executionStats Mode:
        >db.products.explain("executionStats").find(
                { quantity: { $gt: 50 }, category: "apparel" }
        )
        allPlansExecution Mode:
        db.products.explain("allPlansExecution").update(
               { quantity: { $lt: 1000}, category: "apparel" },
```

```
            { $set: { reorder: true } }
        )
```

3.4.12. 重命名集合db.collection.renameCollection(target, dropTarget)

```
>db.rrecord.renameCollection("record")
```

4. 操作符概览

```
Query and Projection Operators
    Query operators provide ways to locate data within the database and
    projection operators modify how data is presented.
Update Operators
    Update operators are operators that enable you to modify the data in
    your database or add additional data.
Aggregation Pipeline Operators
    Aggregation pipeline operations have a collection of operators available
    to define and manipulate documents in pipeline stages.
Query Modifiers
    Query modifiers determine the way that queries will be executed.

Query  Selectors查询选择器

Comparison比较
    For comparison of different BSON type values, see the specified BSON comparison order.
Name    Description
$eq     Matches values that are equal to a specified value.
$gt     Matches values that are greater than a specified value.
$gte    Matches values that are greater than or equal to a specified value.
$lt     Matches values that are less than a specified value.
$lte    Matches values that are less than or equal to a specified value.
$ne     Matches all values that are not equal to a specified value.
$in     Matches any of the values specified in an array.
$nin    Matches none of the values specified in an array.

Logical逻辑运算
Name    Description
$or Joins query clauses with a logical OR returns all documents that
        match the conditions of either clause.
$and    Joins query clauses with a logical AND returns all documents that
        match the conditions of both clauses.
$not    Inverts the effect of a query expression and returns documents that
        do not match the query expression.
$nor    Joins query clauses with a logical NOR returns all documents that fail to match both clauses.
```

```
Element成员操作
Name    Description
$exists Matches documents that have the specified field.
$type   Selects documents if a field is of the specified type.

Evaluation 评估
Name    Description
$mod    Performs a modulo operation on the value of a field and selects documents with a
            specified result.
$regex  Selects documents where values match a specified regular expression.
$text   Performs text search.
$where  Matches documents that satisfy a JavaScript expression.

Array 数组
Name    Description
$all    Matches arrays that contain all elements specified in the query.
$elemMatch  Selects documents if element in the array field matches all the
            specified $elemMatch conditions.
$size   Selects documents if the array field is a specified size.
```

4.1. 操作符详解

4.1.1. $in 属于某个集合

```
$in操作符选择所有满足条件的文档 (文档的一个域等于任意一个给定数组的元素)
使用形式:
    { field: { $in: [<value1>...<valueN> ] } }
    Use the $in Operator to Match Values
$in后面跟上一个条件数组,字段的值为一个数组中任意一个元素的值，并且可以指定不同类型的值

>db.blog.find({user_id:{$in:[2134,"joe"]}})

>db.inventory.find( { qty: { $in: [ 5, 15 ] } } )
Use the $in Operator to Match Values in an Array
>db.inventory.update(
             { tags: { $in: ["appliances", "school"] } },
             { $set: { sale:true } }
           )
更新tags域中只要有application或者school任意其一 sale置为true
否则 不设置sale的值

Use the $in Operator with a Regular Expression
>db.inventory.find( { tags: { $in: [ /^be/, /^st/ ] } } )
```

```
    查找tags域能匹配两个正则表达式的任意之一   即tags以be或者st字符串开头的所有文档
```

4.1.2. $nin 不属于某个集合

```
>db.blog.find({user_id:{$nin:["jone","joe"]}})`
$in是对单个键做OR查询 如果想对多个键做OR查询应该使用$or
$nin返回与条件数组都不匹配的文档
    字段的值不等于数组中任意一个元素的值
```

4.1.3. $exists 指定域存在

```
>db.blog.find({title:{$exists:true}})
查找blog中 title属性存在的文档
{ "_id" : ObjectId("550926214189cd62857147a5"), "title" : "c++ primer" }
```

4.1.4. $mod 取模

```
db.blog.find({age:{"$mod":[5,1]}})
找出age属性能模5余1的所有文档
```

4.1.5. $not 取反

```
和正则表达式联合使用极为有用,用来查找那些与特定模式不匹配的文档
>db.blog.find({age:{$not:{$mod:[5,1]}}})
找出age属性    模5不能余1的所有文档
```

注意：

```
db.blog.find({key: {$not: 2}});
这个是有问题的，如果只是匹配不等于某个值，应该使用：
db.blog.find({key: {$ne: 2}});
db.inventory.find( { price: { $not: { $gt: 1.99 } } } )
查找价格不大于(小于 或等于) 1.99 或者price域不存在
use the $not operator for logical disjunctions and the $ne operator to test
the contents of fields directly
综上，判断一个域的值是否等于指定的值应当用eq 或者 ne
```

4.1.6. $eq 等于

了解更多eq用法 用法:{ field: { $eq: value } } 示例: Equals a Specified Value 等于一个指定的值

db.inventory.find({ qty: { $eq: 20 } }) 查询qty值等于20的文档

```
Field in Embedded Document Equals a Value 嵌套文档的一个域的值等于一个值
    db.inventory.find( { "item.name": { $eq: "ab" } } )
    查找嵌套文档的域item.name 等于ab的文档
Array Element Equals a Value 数组成员等于一个值
    db.inventory.find( { tags: { $eq: "B" } } )
    查找tags数组中有B元素的文档
Equals an Array Value   等于一个数组的值
    The following example queries the inventory collection to select all
    documents where the tags array equals exactly the specified array or
    the tags array contains an element that equals the array ["A", "B"]
    下面的示例要求inventory集合 选择所有满足条件的文档
    需要 tags数组完全等于给定的数组 或者 tags数组包含一个等于给定条件元素
    db.inventory.find( { tags: { $eq: [ "A", "B" ] } } )
    等效于 db.inventory.find( { tags: [ "A", "B" ] } )

    查找tags域含有“A”，“B”两个元素的数组的文档
    { _id: 3, item: { name: "ij", code: "456" }, qty: 25, tags: [ "A", "B" ] }
    { _id: 5, item: { name: "mn", code: "000" }, qty: 20, tags: [ [ "A", "B" ], "C" ] }
```

4.1.7. $ne 不等于

```
Syntax: {field: {$ne: value} }
$ne selects the documents where the value of the field is not equal
 (!=) to the specified value. This includes documents that
      do not contain the field.
$ne操作符选择的文档需要 键值不等于给定的值<这包括文档没有那个域>。
db.inventory.find( { qty: { $ne: 20 } } )
db.inventory.update( { "carrier.state": { $ne: "NY" } },
                     { $set: { qty: 20 } } )
```

4.1.8. $or 或者

```
$in可以用来查询一个键的多个值
$or更通用，查询多个键的给定的任意值
>db.blog.find({$or:{{"name":"joe"},{"winnre":true}}})
The $or operator performs a logical OR operation on an array of two or more
<expressions> and selects the documents that satisfy at least one of the
<expressions> or操作符执行一个逻辑或操作一个数组 或者2个以上的表达式
并且选择选择至少满足一个条件的文档

$or操作符执行逻辑or操作 在一个数组
    { $or: [ { <expression1> },... ,{ <expressionN> } ] }
```

```
>db.inventory.find({$or: [{ quantity:{ $lt: 20 }}, { price: 10 }]} )
    When using $or with <expressions> that are equality checks for the value
    of the same field, use the $in operator instead of the $or operator.
>db.inventory.find ( { quantity:{ $in: [20, 50]}})
```

4.1.9. $and 并且

```
mongoDB中查询条件直接隐含的就是and的关系
>db.inventory.find( { $and: [ { price: { $ne: 1.99 } },{ price: { $exists: true } } ] } )
    同下列等价：
>db.inventory.find( { price: { $ne: 1.99, $exists: true } } )
```

4.1.10. $nor 一个也不

```
$nor用法:{ $nor: [ { <expression1> }, { <expressionN> } ] }
$nor操作符匹配所有条件都为假

>db.blog.find({$nor: [ {key1: value1}, {key2: value2} ] } );
    这个匹配key1不等于value1且key2不等于value2的记录，$nor的多个条件之间是且的关系
```

4.1.11. $type 判断类型

```
$type用法: { field: { $type: <BSON type> } }
$type selects the documents where the value of the field is an instance of the specified numeric
BSON type. This is useful when dealing with highly unstructured data where data types are not
predictable.
选择文档 当键值是一个指定类型的(BSON type)的实例。在处理大量非结构化数据且数据类型不可预测的时候非常有用。
Type        Number  Notes
Double      1
String      2
Object      3
Array       4
Binary data 5
Undefined   6   Deprecated.
Object id   7
Boolean     8
Date        9
Null        10
Regular Expression  11
JavaScript  13
Symbol  14
JavaScript (with scope) 15
32-bit integer  16
Timestamp   17
```

```
64-bit integer  18
Min key 255 Query with -1.
Max key 127

示例代码：
>db.inventory.find( { tags: { $type : 2 } } );
    查找tags的属性是string类型的文档
    更为简单是的可以使用字符串形式
>db.inventory.find({tags:{$type:"string"}});
```

4.1.12. $regex 正则匹配

$regex的更多用法 Provides regular expression capabilities for pattern matching strings in queries.MongoDB uses Perl compatible regular expressions (“PCRE”) version 8.30 with UTF-8 support.

```
{ <field>: { $regex: /pattern/, $options: '<options>' } }
{ <field>: { $regex: 'pattern', $options: '<options>' } }
{ <field>: { $regex: /pattern/<options> } }
如果使用PCRE支持且JS不支持的正则语法需要使用$regex的操作符
$regex 的 $options
```

- i 模式匹配字母不分大小写。
- m 默认情况下，PCRE 认为目标字符串是由单行字符组成的(实际上它可能会包含多行). 如果目标字符串中没有 "\n"字符，或者模式中没有出现“行首”/“行末”字符， 设置这个修饰符不产生任何影响。

  ```
  For patterns that include anchors (i.e. ^ for the start, $ for the end),match at the beginning
  or end of each line for strings with multiline values. Without this option, these anchors
  match at beginning or end of the string.
  ```

- s 模式中的点号元字符匹配所有字符，包含换行符。

  ```
  如果没有这个修饰符，点号不匹配换行符。
  Allows the dot character (i.e. .) to match all characters including newline characters
  ```

- x 如果设置了这个修饰符，模式中的没有经过转义的或不在字符类中的空白数据字符总会被

  ```
  忽略，并且位于一个未转义的字符类外部的#字符和下一个换行符之间的字符也被忽略。
  这个修饰符使被编译模式中可以包含注释。
  注意：这仅用于数据字符。空白字符还是不能在模式的特殊字符序列中出现，比如序列 。
  “Extended” capability to ignore all white space characters in the $regex pattern unless escaped or
  included in a character class.Additionally, it ignores characters in-between and including an
  ```

```
un-escaped hash/pound (#) character and the next new line, so that you may include comments in
complicated patterns. This only applies to data characters; white space characters may never appear
within special character sequences in a pattern.
注：JavaScript只提供了i和m选项，x和s选项必须使用$regex操作符。
执行不区分大小写的正则表达式匹配
```

> db.products.find({ sku: { $regex: /^ABC/i } }) 查找sku域以abc开头部分忽略大小写的文档 多行匹配 db.products.find({ description: { $regex: /^S/, $options: 'm' } }) { *"id":100, "sku" : "abc123", "description" : "Single line description." } { "*id":101, "sku" : "abc789", "description" : "First line\nSecond line" } 若没有m选项

```
{ "_id":100,"sku":"abc123", "description" : "Single line description." }
```

Use the . Dot Character to Match New Line使用.点字符 用来匹配新字符

> db.products.find({ description: { $regex: /m.*line/, $options: 'si' } }) { *"id":102,"sku":"xyz456", "description": "Many spaces before line" } { "*id":103,"sku":"xyz789","description" : "Multiple\nline description" } 若忽略s选项 { "_id":102,"sku":"xyz456","description": "Many spaces before line" }

Ignore White Spaces in Pattern 忽略空白字符

> var pattern = "abc #category code\n123 #item number" db.products.find({ sku: { $regex: pattern, $options: "x" } }) 查找 忽略# \n之间的字符并且能匹配的文档

4.1.13. $elemMatch 成员匹配

```
The $elemMatch operator matches documents that contain an array field with at least one element
that matches all the specified query criteria.
使用结构:{ <field>: { $elemMatch: { <query1>, <queryn>} } }

{ _id: 1, results: [ 82, 85, 88 ] }
{ _id: 2, results: [ 75, 88, 89 ] }

#查找results数组中至少有一个元素满足条件 >=80 && <85 的文档
>db.scores.find({ results: { $elemMatch: { $gte: 80, $lt: 85 } } })
{ "_id" : 1, "results" : [ 82, 85, 88 ] }

{ _id: 1, results: [ { product: "abc", score: 10 }, { product: "xyz", score: 5 } ] }
{ _id: 2, results: [ { product: "abc", score: 8 }, { product: "xyz", score: 7 } ] }
```

```
{ _id: 3, results: [ { product: "abc", score: 7 }, { product: "xyz", score: 8 } ] }

>db.survey.find({results:{ $elemMatch:{ product:"xyz",score:{$gte: 8}}}})
{ _id: 3, results:[{ product: "abc", score:7 },{ product:"xyz",score:8}]}

查询文档有两种方式，一种是完全匹查询，另一种是针对键/值对查询。
内嵌文档的完全匹配查询和数组的完全匹配查询一样，内嵌文档内键值对的数量，顺序都必须一致才会匹配.
通过点表示法来精确表示内嵌文档的键
当内嵌文档变得复杂后，如键的值为内嵌文档的数组，内嵌文档的匹配需要些许技巧

例如使用$elemMatch操作符
{
    "content" : ".....",
    "comment" : [{"author" : "zhangsan","score": 3,"comment" : "shafa!"},
                 {"author" : "lisi","score" : 5,"comment" : "lzsb!"}
                ]
}
查询评论中用户“zhangsan”是否有评分超过4分的评论内容>
>db.blogs.find({"comment.author":"zhangsan", "comment.score":{"$gte":4}});
结果不对

下面使用“$elemMatch”操作符即可将一组条件限定到数组中单条文档的匹配上
$elemMatch with Multiple Fields匹配多个域

>db.blog.find({"comment":{"$elemMatch":{author:"zhangsan",score:{$gt:4}}}});
>db.schools.find( { zipcode: "63109"},{students: { $elemMatch: { school: 102, age: { $gt: 10}}}})
返回的结果:

    { "_id" : 1, "students" :[ { "name": "jess", "school" : 102, "age": 11}]}
    { "_id" : 3 }
    { "_id" : 4, "students" :[ { "name": "ruth", "school" : 102, "age": 16}]}
```

4.1.14. $all 全部

```
匹配那些指定键的键值中包含数组，而且该数组包含条件指定数组的所有元素的文档，
数组中元素顺序不影响查询结果
使用结构:{ <field>: { $all: [ <value1> , <value2> ... ] } }
$all后面的数组的每个条件之间是  and 且的关系，与顺序无关

#   找在集合blog中 tags键值包含数组，且数组中包含 含有appliances,school,book元素的文档:
>db.blog.find( { tags: { $all: [ "appliances", "school", "book" ] } } )

#$all与$elemMatch
>db.inventory.find( {
                qty:{ $all: [
```

```
                            { "$elemMatch" : { size: "M", num: { $gt: 50}} },
                            { "$elemMatch" : { num : 100, color: "green"} }
                             ] }
                  } )
查询qty是数组，且数组中含有 (size元素为M num>50)  且  (num等于100 颜色为绿色)条件的元素    的文档
```

4.1.15. $size 数组大小

```
用其查询指定长度的数组
>db.collection.find( { field: { $size: 2 } } );

returns all documents in collection where field is an array with 2 elements.
返回所有在collection集合中有两个元素的数组的 文档

$size不接受范围。选择文档基于字段与不同数量的元素,创建一个计数器字段,你添加元素时增加一个那个字段的值。
Queries cannot use indexes for the $size portion of a query, although the other portions of a query
can use indexes if applicable.
$size部分 的查询不能使用索引,虽然其他部分     如果适用可以使用索引.

###$text
###$where
```

4.2. Projection Operators 映射操作符

```
Name        Description
$           Projects the first element in an array that
                matches the query condition.
$elemMatch  Projects the first element in an array that
                matches the specified $elemMatch condition.
$meta       Projects the document’s score assigned during $text operation.
$slice      Limits the number of elements projected from an array.
                Supports skip and limit slices.
```

```
示例：
```

4.2.1. $ 指定下标

```
The positional $ operator limits the contents of an <array> from the query results to contain
only the first element matching the query document.
$符限制  <数组>的内容从查询结果只包含第一个元素匹配查询文档。
使用$在文档中的find方法或者findOne方法,当你只需要一个特定的数组元素在选定的文档
使用结构
```

```
    db.collection.find( { <array>: <value> ... },{ "<array>.$": 1 } )
    db.collection.find( { <array.field>: <value> ...},{ "<array>.$": 1 } )

The <array> field being limited must appear in the query document, and the <value> can be
documents that contain query operator expressions.
数组field被限制必须出现在查询文档,并且field的值也可以是包含查询操作符表达式的文档

Array Field Limitations数组域的限制
MongoDB requires the following when dealing with projection over arrays:
需要满足以下条件
Only one positional $ operator may appear in the projection document.
只有一个$能出现在文档中

Only one array field may appear in the query document.
只有一个数组域能够出现查询文档中
The query document should only contain a single condition on the array field being projected.
Multiple conditions may override each other internally and lead to undefined behavior.
查询文档只能包含单一的条件在数组域上;多条件会覆盖其他,导致不可预期的行为.

You must use the $elemMatch operator if you need separate conditions for
selecting documents and for choosing fields within those documents.
你需要使用$elemMathc符号 如果你需要分条件为了查找文档 选定这些文档的一些域

    { "_id" : 3, "semester" : 1, "grades" : [ 85, 100, 90 ] }
>db.students.find( { semester: 1, grades: { $gte: 85 } },{ "grades.$": 1 }))
查找semester为1 且 分数大于等于85 且放回第一科分数
    { "_id" : 3, "grades" : [ 85 ] }

> db.blog.find({"grades.mean":{$gte:90}},{"grades.$":1})
查找grades中mean域大于等于90的数组 并且返回 该文档中数组元素的第一个元素
    { "_id" : 7, "grades" : [ { "grade" : 85, "mean" : 90, "std" : 5 } ] }
    { "_id" : 8, "grades" : [ { "grade" : 78, "mean" : 90, "std" : 5 } ] }
```

4.2.2. $slice 取出元素

```
The $slice operator controls the number of items of an array that a query returns. For information
on limiting the size of an array during an update with $push, see the $slice modifier instead
$slice操作符控制查询 返回的数组元素个数

>db.collection.find( { field: value }, {array:{$slice: count }});
如果count的值大于数组中元素的数量，该查询返回数组中的所有元素的

选择comments的数组键值中前五个元素。
>db.posts.find( {}, { comments: { $slice: 5 } } );
```

```
选择comments的数组键值中后五个元素。
>db.posts.find( {}, { comments: { $slice: -5 } } );

数组参数使用[ skip , limit ] 格式，
其中第一个值表示在数组中跳过的项目数,第二个值表示返回的项目数
>db.blog.find({},comm:{$slice:[20,10]})
    跳过前20项从前取10个
>db.blog.find({},comm:{$slice:[-20,10]})
    跳过后20项从后取10个
```

4.2.3. $elemMatch

```
The $elemMatch operator limits the contents of an <array> field from the query results to contain only
the first element matching the $elemMatch condition
操作符限制 从数组域中查询的结果集的内容 （仅包含第一个匹配$elemMatch条件的成员）
    {
     _id: 1,
     zipcode: "63109",
     students: [
                   { name: "john", school: 102, age: 10 },
                   { name: "jess", school: 102, age: 11 },
                   { name: "jeff", school: 108, age: 15 }
                ]
    }
    {
     _id: 2,
     zipcode: "63110",
     students: [
                   { name: "ajax", school: 100, age: 7 },
                   { name: "achilles", school: 100, age: 8 },
                ]
    }
    {
     _id: 3,
     zipcode: "63109",
     students: [
                   { name: "ajax", school: 100, age: 7 },
                   { name: "achilles", school: 100, age: 8 },
                ]
    }
    {
     _id: 4,
     zipcode: "63109",
     students: [
                   { name: "barney", school: 102, age: 7 },
```

```
                { name: "ruth", school: 102, age: 16 },
              ]
    }
>db.schools.find( { zipcode: "63109" },{ students: { $elemMatch: { school: 102 } } } )
{ "_id" : 1, "students" : [ { "name" : "john", "school" : 102, "age" : 10 } ] }
{ "_id" : 3 }
{ "_id" : 4, "students" : [ { "name" : "barney", "school" : 102, "age" : 7 } ] }

$elemMatch 用于多个域的匹配

>db.schools.find({zipcode:"63109"},
               {student:{$elemMatch:{school:102,age:{$gt:10}}}})
```

4.3. Update Operators 更新操作符

```
The following modifiers are available for use in update operations; e.g.
in db.collection.update() and db.collection.findAndModify()
下列修改器被使用域更新操作时有效，db.collection.update() or db.collection.findAndModify()

$inc                    指定键  增加(或者减少)指定值
$set                    指定键 修改值(没有则新建)
$unset                  删除键值

$       定位操作符（"$"）.数组是0开始的，可以直接将下标作为键来选择元素。
        作为一个占位符来更新查询条件相匹配的第一个元素在一个更新。
$mul    将匹配的域对应的值乘上指定的值.

$min    Only updates the field if the specified value is less than the existing field value.
             只修改对应的域(指定那个值  小于 那个存在的域的值)
$max    Only updates the field if the specified value is greater than the existing field value.
             只修改对应的域 如果指定的值大于 那个存在的域的值
$addToSet   Adds elements to an array only if they do not already exist in the set.
                添加成员到一个数组 只有当这个元素在文档集中不存在时(集合的一个性质就是值不重复)
$pop        Removes the first or last item of an array.移除 第一个[1] 倒数第一个[-1]元素从一个数组中。

$pullAll    Removes all matching values from an array.
            从数组中 移除所有匹配的值
$pull       Removes all array elements that match a specified query.
                移除数组中所有  匹配一个指定查询的成员
$pushAll    Deprecated. Adds several items to an array.
                弃用。添加一些项到一个数组中
$push       Adds an item to an array.
                添加一项到一个数组中
$each       Modifies the $push and $addToSet operators to append multiple items for array updates.
```

```
$slice      Modifies the $push operator to limit the size of updated arrays.
$sort       Modifies the $push operator to reorder documents stored in an array.
$position   Modifies the $push operator to specify the position in the array to add elements.

$rename     Renames a field.    重命名一个域。
$currentDate    Sets the value of a field to current date, either
                as a Date or a Timestamp.
$setOnInsert    Sets the value of a field if an update results in an insert
                of a document. Has no effect on update operations
                that modify existing documents.
```

4.3.1. $inc 增加指定域的值

```
    $inc 加上一个指定的值
    使用结构:
        { $inc: { <field1>: <amount1>, <field2>: <amount2>, ... } }
The $inc operator accepts positive and negative values.
If the field does not exist, $inc creates the field and sets the field to the specified value.
$inc is an atomic operation within a single document.
$inc操作结构一个正数值 和 负数值。如果指定域不存在，$inc创建指定的域并且将值置为指定的值。
$inc是在一个原子性操作。

> db.blog.update({title:"test"},{$inc:{age:1}})
    WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.blog.find()
    { "_id" : ObjectId("5503ecd83853a67e204962ae"), "title1" : "instrduce of mongo" }
    { "_id" : ObjectId("5503ed873853a67e204962af"), "docment" : "hello mongoDB!" }
    { "_id" : ObjectId("5503edee3853a67e204962b0"), "title" : "mytest", "name" : "pc" }
    { "_id" : ObjectId("5507f384b63145a5ffc5a59f"), "title" : "book1", "comment" : "bad", "count" : Number
    { "_id" : ObjectId("5507edd4b63145a5ffc5a59e"), "title" : "book", "comment" : "good", "relation_array"
    { "_id" : ObjectId("550926214189cd62857147a5"), "title" : "test", "age" : 35 }
    { "_id" : ObjectId("550926334189cd62857147a6"), "title" : "test", "age" : 34 }
> db.blog.update({title:"test"},{$inc:{age:1}},false,true)
    WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })
> db.blog.find()
    { "_id" : ObjectId("5503ecd83853a67e204962ae"), "title1" : "instrduce of mongo" }
    { "_id" : ObjectId("5503ed873853a67e204962af"), "docment" : "hello mongoDB!" }
    { "_id" : ObjectId("5503edee3853a67e204962b0"), "title" : "mytest", "name" : "pc" }
    { "_id" : ObjectId("5507f384b63145a5ffc5a59f"), "title" : "book1", "comment" : "bad", "count" : Number
    { "_id" : ObjectId("5507edd4b63145a5ffc5a59e"), "title" : "book", "comment" : "good", "relation_array"
    { "_id" : ObjectId("550926214189cd62857147a5"), "title" : "test", "age" : 36 }
    { "_id" : ObjectId("550926334189cd62857147a6"), "title" : "test", "age" : 35 }
    { "_id" : ObjectId("5507edd4b63145a5ffc5a59e"), "title" : "book", "comment" : "good", "relation_array"

>db.blog.update({"title":"book"},{$push:{relation_array:bad}}})
```

```
    { "_id" : ObjectId("5507edd4b63145a5ffc5a59e"), "title" : "book", "comment" : "good", "relation_array"
    { "_id" : ObjectId("550926214189cd62857147a5"), "title" : "test", "age" : 36 }
> db.blog.update({"title":"test"},{title:"t"})`
    { "_id" : ObjectId("550926214189cd62857147a5"), "title" : "t" }
> db.blog.update({"title":"test"},{$set:{title:"t"}})
    { "_id" : ObjectId("550926334189cd62857147a6"), "title" : "t", "age" : 35 }
```

4.3.2. $mul 乘上指定的值

```
{ $mul: { field: <number> } }
If the field does not exist in a document, $mul creates the field and sets the value to
zero of the same numeric type as the multiplier.
原子性操作。
如果需要匹配的域在文档中不存在,$mul创建那个域并且设置成与number相同的类型的0.
{ _id: 1, item: "ABC", price: 10.99 }
>db.blog.update({_id:1},{$mul:{price,10}})
{ _id: 1, item: "ABC", price: 109.9 }
```

```
涉及到类型不同的时候有一个隐含的类型转换.

            NumberInt(integer32) NumberLong(integer64)      NumberDouble()
32-bit Integer      32 or Integer64 Integer64               Float
64-bit Integer      Integer64       Integer64               Float
        Float       Float           Float                   Float
```

4.3.3. $rename 重命名

```
对指定field重命名成newName。
    {$rename: { <field1>: <newName1>, <field2>: <newName2>, ... } }
逻辑上执行$rename操作的时候，首先$unset field 和newname,然后进行$set newname操作。

>db.students.update( { _id: 1 }, { $rename:{"name.first": "name.fname"}})
    {"_id": 1,"name" : { "fname" : "george", "last" : "washington" } }
```

4.3.4. $setOnInsert 匹配则设置

```
使用形式：
    db.collection.update(
       <query>,
       { $setOnInsert: { <field1>: <value1>, ... } },
       { upsert: true }
    )
```

```
If the db.collection.update() with upsert:true had found a matching document,then MongoDB performs
an update, applying the $set operation but ignoring the $setOnInsert operation.
upsert为true 且找到了匹配文档 将执行更新 应用$set操作 但是忽略$setOnInsert操作。

If the update operation does not result in an insert,$setOnInsert does nothing.
如果更新操作没有结果
products没有文档
$setOnInsert操作符只是在指定域创建的时候设置对应的值,否则不设置.

>db.products.update({ _id: 1 },{ $set: { item: "apple" },
                           $setOnInsert: { defaultQty: 100 }
                         },{upsert: true }
                    )
    { "_id" : 1, "item" : "apple", "defaultQty" : 100 }
```

4.3.5. $set 设置指定的值

```
The $set operator replaces the value of a field with the specified value.
$set操作符会用指定的值去替换指定域的值。
If you specify multiple field-value pairs, $set will update or create each field.
如果你指定了多个键值对，$set将会更新或创建每个域

> db.blog.update({"title":"book"},{$set:{"relation_array":["very good",3.14,NumberLong(23)]}});
> db.blog.find()
{ "_id" : ObjectId("5503ecd83853a67e204962ae"), "title1" : "instrduce of mongo" }
{ "_id" : ObjectId("5503ed873853a67e204962af"), "docment" : "hello mongoDB!" }
{ "_id" : ObjectId("5503edee3853a67e204962b0"), "title" : "mytest", "name" : "pc" }
{ "_id" : ObjectId("5507f384b63145a5ffc5a59f"), "title" : "book1", "comment" : "bad" }
{ "_id" : ObjectId("5507edd4b63145a5ffc5a59e"), "title" : "book", "comment" : "good",
    "relation_array" : [ "very good", 3.14, NumberLong(23) ] }
> db.blog.find()
    { "_id" : ObjectId("5503ecd83853a67e204962ae"), "title1" : "instrduce of mongo" }
    { "_id" : ObjectId("5503ed873853a67e204962af"), "docment" : "hello mongoDB!" }
    { "_id" : ObjectId("5503edee3853a67e204962b0"), "title" : "mytest", "name" : "pc" }
    { "_id" : ObjectId("5507f384b63145a5ffc5a59f"), "title" : "book1", "comment" :
        "bad", "count" : NumberLong(123), "size" : NumberLong(123) }
    { "_id" : ObjectId("5507edd4b63145a5ffc5a59e"), "title" : "book", "comment" : "good",
        "relation_array" : [ "very good", 3.14, NumberLong(23) ] }
    { "_id" : ObjectId("550926214189cd62857147a5"), "title" : "test", "age" : 32 }
    { "_id" : ObjectId("550926334189cd62857147a6"), "title" : "test", "age" : 33 }
> db.blog.update({title:"test"},{$inc:{age:1}})
    WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.blog.find()
    { "_id" : ObjectId("5503ecd83853a67e204962ae"), "title1" : "instrduce of mongo" }
    { "_id" : ObjectId("5503ed873853a67e204962af"), "docment" : "hello mongoDB!" }
```

```
    { "_id" : ObjectId("5503edee3853a67e204962b0"), "title" : "mytest", "name" : "pc" }
    { "_id" : ObjectId("5507f384b63145a5ffc5a59f"), "title" : "book1", "comment" : "bad", "count" :
                NumberLong(123), "size" : NumberLong(123) }
    { "_id" : ObjectId("5507edd4b63145a5ffc5a59e"), "title" : "book", "comment" :
                "good", "relation_array" : [ "very good", 3.14, NumberLong(23) ] }
    { "_id" : ObjectId("550926214189cd62857147a5"), "title" : "test", "age" : 33 }
    { "_id" : ObjectId("550926334189cd62857147a6"), "title" : "test", "age" : 33 }
> db.blog.update({title:"test"},{$inc:{age:1}},false,true)
    WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })
> db.blog.find()
    { "_id" : ObjectId("5503ecd83853a67e204962ae"), "title1" : "instrduce of mongo" }
    { "_id" : ObjectId("5503ed873853a67e204962af"), "docment" : "hello mongoDB!" }
    { "_id" : ObjectId("5503edee3853a67e204962b0"), "title" : "mytest", "name" : "pc" }
    { "_id" : ObjectId("5507f384b63145a5ffc5a59f"), "title" : "book1", "comment" : "bad", "count" :
            NumberLong(123), "size" : NumberLong(123) }
    { "_id" : ObjectId("5507edd4b63145a5ffc5a59e"), "title" : "book", "comment" :
            "good", "relation_array" : [ "very good", 3.14, NumberLong(23) ] }
    { "_id" : ObjectId("550926214189cd62857147a5"), "title" : "test", "age" : 34 }
    { "_id" : ObjectId("550926334189cd62857147a6"), "title" : "test", "age" : 34 }
```

4.3.6. $unset 删除指定域

```
$unset操作删除一个指定的域，使用形式：
    { $unset: { <field1>: "", ... } }
If the field does not exist, then $unset does nothing (i.e. no operation).
如果指定域不存在，$unset操作什么也不做。
    { "_id" : ObjectId("553f6d670d426b8bc8331730"), "name" : "it", "age" : 22, "wife" : "bug" }
> db.blog.update({name:"it"},{$unset:{wife:1}})
    WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.blog.find()
    { "_id" : ObjectId("553f6d670d426b8bc8331730"), "name" : "it", "age" : 22 }
> db.blog.update({name:"it"},{$unset:{age:""}})
    WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.blog.find()
    { "_id" : ObjectId("553f6d670d426b8bc8331730"), "name" : "it" }
```

4.3.7. $min

```
使用形式:
{ $min: { <field1>: <value1>, ... } }
If the field does not exists, the $min operator sets the field to the specified value.The $min updates
the value of the field to a specified value if the specified value is less than the current value of
the field.
如果指定的域不存在，$min操作将设置指定域为指定的值。
```

```
$min操作符更新 文档的域（比指定值大）  为指定的值。
    { _id: 1, highScore: 800, lowScore: 200 }
>db.scores.update( { _id: 1 }, { $min: { lowScore: 150 } } )
    { _id: 1, highScore: 800, lowScore: 150 }

>db.scores.update( { _id: 1 }, { $min: { lowScore: 250 } } )
    { _id: 1, highScore: 800, lowScore: 150 }
```

4.3.8. $max

```
If the field does not exists, the $max operator sets the field to the specified value.
如果域不存在 将会设置对应的域为指定的值
The $max operator updates the value of the field to a specified value if the specified value
is greater than the current value of the field.
$max操作符更新 文档的域 比指定值小的域
    { _id: 1, highScore: 800, lowScore: 200 }
>db.scores.update( { _id: 1 }, { $max: { highScore: 950 } } )
    { _id: 1, highScore: 950, lowScore: 200 }
>db.scores.update( { _id: 1 }, { $max: { highScore: 780 } } )
    { _id: 1, highScore: 950, lowScore: 200 }
```

4.3.9. $currentDate

```
The $currentDate operator sets the value of a field to the current date, either as a Date or
a timestamp. The default type is Date.
设置 一个域的值为当前的时间 要么是日期<默认> 要么是时间戳.
Changed in version 3.0: MongoDB no longer treats the timestamp and the Date data types
as equivalent for comparison/sorting purposes.
#3.0中不再将timstamp和日期看成等效在比较和排序中
如果对应的域没有, $currentDate adds the field to a document.

使用结构:
{ $currentDate: { <field1>: <typeSpecification1>, ... } }
typeSpecification为{ $type: "timestamp" } or { $type: "date" }

If the field does not exist, $currentDate adds the field to a document.
如果域不存在，$currentData添加一个域到文档中。

{ _id: 1, status: "a", lastModified: ISODate("2013-10-02T01:11:18.965Z") }
>db.users.update(
                   { _id: 1 },
                   {
                     $currentDate: {
                        lastModified: true,
                        "cancellation.date": { $type: "timestamp" }
```

```
                    },
                    $set: {
                       status: "D",
                       "cancellation.reason": "user request"
                    }
                 }
              )
{   "_id" : 1, "status" : "D", "lastModified" : ISODate("2014-09-17T23:25:56.314Z"),
    "cancellation" : { "date" : Timestamp(1410996356, 1),
    "reason" : "user request"}
}
```

4.3.10. $ 占位操作符

```
使用格式1:  更新数组中的值
{ "<array>.$" : value }

db.collection.update({ <array>: value ... },
                     { <update operator>: { "<array>.$" : value } }
                   )
Remember that the positional $ operator acts as a placeholder for the first match of the update query
document.记住 位置操作符$ 作为一个占位符
> db.blogs.find()
    { "_id" : 1, "grades" : [ 80, 85, 90 ] }
    { "_id" : 2, "grades" : [ 88, 90, 92 ] }
    { "_id" : 3, "grades" : [ 85, 100, 90 ] }
> db.blogs.update(    { _id: 1,grades:80},{ $set: { "grades.$" : 82 } } )
    WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.blogs.find()
    { "_id" : 1, "grades" : [ 82, 85, 90 ] }
    { "_id" : 2, "grades" : [ 88, 90, 92 ] }
    { "_id" : 3, "grades" : [ 85, 100, 90 ] }
> db.blogs.update(    { _id: 1,grades:85},{ $set: { "grades.$" : 82 } } )
    WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.blogs.find()
    { "_id" : 1, "grades" : [ 82, 82, 90 ] }
    { "_id" : 2, "grades" : [ 88, 90, 92 ] }
    { "_id" : 3, "grades" : [ 85, 100, 90 ] }

使用格式2:  更新数组中的文档
#db.collection.update(
        { <query selector> },
        { <update operator>: { "array.$.field" : value } }
        )
    {
```

```
      _id: 4,
      grades: [
         { grade: 80, mean: 75, std: 8 },
         { grade: 85, mean: 90, std: 5 },
         { grade: 90, mean: 85, std: 3 }
      ]
    }
    db.students.update( { _id: 4, "grades.grade": 85 },
                        { $set: { "grades.$.std" : 6 } }
                      )
使用格式3: 多键值匹配更新嵌套文档
    >db.students.update(
    {
        _id: 4,
        grades: { $elemMatch: { grade: { $lte: 90 }, mean: { $gt: 80 } } }
    },
    { $set: { "grades.$.std" : 6 } }
    )
        {
            _id: 4,
            grades: [
                { grade: 80, mean: 75, std: 8 },
                { grade: 85, mean: 90, std: 6 },
                { grade: 90, mean: 85, std: 3 }
            ]
        }
```

4.3.11. $addToSet 添加到集合

```
The $addToSet operator adds a value to an array only if the value is not already in the array.
If the value is in the array,$addToSet does not modify the array.
添加一个值到一个数组中 如果数组中原来存在就添加 否则不添加
使用格式:
    { $addToSet: { <field1>: <value1>, ... } }

1.  $addToSet只保证没有重复的条目添加到域中和不影响现有重复的元素.但是,并不保证被修改集合的元素间的顺序.
2.  如果需要更新的域不存在,$addToSet创建数组域并将制定的值作为数组的元素.
3.  如果需要更新的域不是数组类型,操作就会失败
4.  如果指定的是一个数组类型,指定的这个数组就会作为更新的数组的一个元素
5.  如果想把数组中的每个元素加入到目标数组，就使用$each取出指定数组元素,加入到数组中。
    { _id: 1, letters: ["a", "b"] }
>db.test.update({ _id: 1},{ $addToSet: {letters:{$each: ["c", "d"]}}})
    { _id: 1, letters: [ "a", "b", "c", "d"] }

    { _id: 1, letters: ["a", "b"] }
```

```
>db.test.update({ _id: 1 },{ $addToSet: {letters: [ "c", "d" ] } })
    { _id: 1, letters: [ "a", "b", [ "c", "d" ] ] }
```

4.3.12. $pop 移除指定元素

```
The $pop operator removes the first(-1) or last(1) element of an array.
从数组中移除第一个或者最后一个成员。
如果指定的域不是数组将会失败。
If the $pop operator removes the last item in the <field>, the <field> will then hold an empty array.
如果$pop操作移除了指定域中最后一项，那么指定域将会变成一个空数组。

使用格式:   { $pop: { <field>: <-1 | 1>, ... } }

    { "_id" : 1, "letters" : [ "a", "b" ], "letter" : [ [ "c", "d" ], "c", "d" ] }
> db.blogs.update( {_id:1},{$pop:{letter:-1}} )
    WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.blogs.find()
    { "_id" : 1, "letters" : [ "a", "b" ], "letter" : [ "c", "d" ] }
> db.blogs.update({_id:1},{$pop:{letter:1}})
    WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.blogs.find()
    { "_id" : 1, "letters" : [ "a", "b" ], "letter" : [ "c" ] }
```

4.3.13. $pullAll

```
removes all instances of the specified values from an existing array.
    从已经存在的数组中移除所有列表中匹配的元素.
$pullAll removes elements that match the listed values
使用格式:{ $pullAll: { <field1>: [ <value1>, <value2> ... ], ... } }

    { _id: 1, scores: [ 0, 2, 5, 5, 1, 0 ] }
>db.survey.update( { _id: 1 }, { $pullAll: { scores: [ 0, 5 ] } } )
    { "_id" : 1, "scores" : [ 2, 1 ] }
```

4.3.14. $pull

```
使用格式:{ $pull: { <field1>: <value|query>, ... } }
If the specified <value> to remove is an array,$pull removes only the elements in the array
that match the specified <value> exactly,including order.
If the specified <value> to remove is a document, $pull removes only the elements in the array
that have the exact same fields and values.
如果指定的要删除的值是一个数组，$pop将仅移除 完全匹配指定值的 包括顺序。

The ordering of the fields can differ
```

```
    { _id: 1, flags: [ "vme", "de", "msr", "tsc", "pse", "msr" ] }
    { _id: 2, flags: [ "msr", "pse", "tsc" ] }
>db.cpuinfo.update({flags: "msr"},{ $pull: {flags: "msr"}},{ multi:true })
    { _id: 1, flags: [  "vme",  "de",  "tsc",  "pse" ] }
    { _id: 2, flags: [  "pse",  "tsc" ] }

    { _id: 1, votes: [ 3, 5, 6, 7, 7, 8 ] }
>db.cpuinfo.update({_id:1},{$pull:{votes:{gte:6}}})
    { _id: 1, votes: [ 3, 5] }

##思考题  $pop与$popAll的区别是啥?
```

4.3.15. $pushAll

```
使用格式:{ $pushAll: { <field>: [ <value1>, <value2>, ... ] } }
eprecated since version 2.4: Use the $push operator with $each instead.
已经被弃用，使用$push操作符加上$each联合使用替代。
The $pushAll operator appends the specified values to an array.
```

4.3.16. $push

```
The $push operator appends a specified value to an array.追加一个指定的值到数组中.
使用形式:
{ $push: { <field1>: <value1>, ... }
If the field is not an array, the operation will fail.不是数组就失败
如果指定的域在文档中没有，$push将会追加整个数组作为一个单一的成员。
如果需要添加每一个元素到指定数组，需要结合使用$each修改器。
>db.students.update({ name: "joe" },
               { $push: { scores: { $each: [ 90, 92, 85 ] } } }
              )

##思考题  $push与$pushAll的区别是啥?
更多$push的用法。
http://docs.mongodb.org/manual/reference/operator/update/push/#up._S_push
```

```
下列的操作符Update Operator Modifiers，更新操作修改器。
均可以与$push操作符共用。
```

4.3.17. $each 追加多个值到数组

```
The $each modifier is available for use with the $addToSet operator and the $push operator.
使用$each   和$addToSet $push合用有效
```

```
Use with the $addToSet operator to add multiple values to an array <field> if the values do not exist
in the <field>.
在指定域中的值不存在 和$addToSet合用 将可以添加多个值到数组域中.

使用形式:
        { $addToSet: { <field>: { $each: [ <value1>, <value2> ... ] } } }
Use with the $push operator to append multiple values to an array <field>.
和$push合用将追加多个值到数组域中.

使用形式:   { $push: { <field>: { $each: [ <value1>, <value2> ... ] } } }
> db.blogs.update({_id:1},{"$push":{grades:{"$each":[12,123,343,433,433]}}})
    { "_id" : 1, "grades" : [ 82, 82, 90 ] }
> db.blogs.update({_id:1},{"$addToSet":{grades:"$each":12,123,343,433,433]}}})
    { "_id" : 1, "grades" : [ 82, 82, 90, 12, 123, 343, 433, 433 ] }
> db.blogs.update({_id:1},{"$addToSet":{grades:{"$each":[122,123,122]}}})
    { "_id" : 1, "grades" : [ 82, 82, 90, 12, 123, 343, 433, 433, 122 ] }
```

4.3.18. $slice

```
The $slice modifier limits the number of array elements during a $push operation.
修改器限制数组成员的个数在执行$push操作期间

使用格式:
{
    $push:   {
    <field>: {
    $each: [ <value1>, <value2>, ... ],
        $slice: <num>
        }
    }
}

Value       Description

Zero        To update the array <field> to an empty array.
Negative    To update the array <field> to contain only the last <num> elements.
Positive    To update the array <field> contain only the first <num> elements.

如果<num> 等于0,清空数组;大于0,保留前num个数;小于0 保留后num个数.

Trying to use the $slice modifier without the $each modifier results in an error.
如果抛弃了$each,单独的$slice修改器运行的结果就会报错.
To use the $slice modifier, it must appear with the $each modifier. You can pass an empty array []
to the $each modifier such that only the $slice modifier has an effect.
使用$slice修改器,必须和$each修改器一起出现.你可以使用$each操作空数组,可以让$slice修改器生效.
```

```
示例:
>db.students.update(  { _id: 3 },
                      { $push: { scores: {$each: [ ],$slice: -3}}
                      })
```

4.3.19. $sort

```
http://docs.mongodb.org/manual/reference/operator/update/sort/

The $sort modifier orders the elements of an array during a $push operation.
$sort修改器 在$push操作期间 对数组的成员进行排序
To use the $sort modifier, it must appear with the $each modifier.You can pass an empty array []
to the $each modifier such that only the $sort modifier has an effect.
$sort修改器必须和$each一起出现,也同使用一个空数组[],让$sort修改器有效果。

使用格式:
{
    $push: {
    <field>: {
    $each: [ <value1>, <value2>, ... ],
    $sort: <sort specification>
             }
           }
}
For <sort specification>:
To sort array elements that are not documents, or if the array elements are documents,
to sort by the whole documents, specify 1 for ascending or -1 for descending.

If the array elements are documents, to sort by a field in the documents, specify a sort document
with the field and the direction, i.e. { field: 1 } or { field: -1 }.
为了排序数组成员(不是文档 或者是文档)，给整个文档排序，指定1升序，-1降序。

$sort modifier can sort array elements that are not documents.In previous versions,the $sort modifier
required the array elements be documents.
$sort修改器能够对成员不是文档的成员排序,在之前的版本中成员须是文档.

If the array elements are documents, the modifier can sort by either the whole document or
by a specific field in the documents.In previous versions,the $sort modifier can only sort by
a specific field in the documents.
如果数组成员是文档,修改器能够对整个文档或者指定域排序,以前只能针对指定域排序.
Trying to use the $sort modifier without the $each modifier results in an error.
The $sort no longer requires the $slice modifier
使用$sort修改而不带$each修改器的结果是错误.$sort不再需要$slice修改器.
数组中的元素是文档
```

```
    { "_id": 1,"quizzes": [
        { "id" : 1, "score" : 6 },
        { "id" : 2, "score" : 9 }
                          ]
    }
>db.blogs.update(   {_id:1},{ $push:{
                    quizzes:{$each: [ { id: 3, score: 8 },{ id: 4, score: 7 }, { id: 5, score: 6 } ],
                             $sort:{score:1}
                            }
                    }
                )
The sort document refers directly to the field in the documents and does not reference the containing
array field quizzes;{ score: 1 } and NOT { "quizzes.score": 1}
排序文档直接引用在文档中的域,而不需要包括数组域引用.
{
    "_id" : 1,
    "quizzes" : [
    { "id" : 1, "score" : 6 },
    { "id" : 5, "score" : 6 },
    { "id" : 4, "score" : 7 },
    { "id" : 3, "score" : 8 },
    { "id" : 2, "score" : 9 }
                ]
}
数组中的元素不是文档
    { "_id" : 2, "tests" : [  89,  70,  89,  50 ] }
>db.students.update({ _id: 2 },{
        $push: { tests: { $each: [ 40, 60 ], $sort: 1 } } }
              )
    { "_id" : 2, "tests" : [  40,  50,  60,  70,  89,  89 ] }
不插入新元素对数组中原有元素排序
>db.students.update({ _id: 3 },{ $push:{tests: { $each:[ ], $sort: -1 }}})

>db.students.update( { _id: 5 },{ $push: {
                      quizzes: {
                      $each: [ { wk: 5, score: 8 }, { wk: 6, score: 7 }, { wk: 7, score: 6 } ],
                      $sort: { score: -1 },
                      $slice: 3
                               }
                     }})
$each将数组中的文档添加到quizzes数组中,降序排序,限制数组长度为3个.
```

4.3.20. $position

```
The $position modifier specifies the location in the array at which the $push operator insert elements.
```

```
Without the $position modifier, the $push operator inserts elements to the end of the array.
$position修改器指定$push操作符插入成员在数组中位置.如果不带$position,数组后追加.

To use the $position modifier, it must appear with the $each modifier
为了使用$position修改器，必须和$each修改器一起出现。
基本使用结构:
{
  $push: {
    <field>: {
       $each: [ <value1>, <value2>, ... ],
       $position: <num>
    }
  }
}
The <num> is a non-negative number that corresponds to the position in the array, based on a zero-based
index.If the <num> is greater or equal to the length of the array, the $position modifier has no effect
and $push adds elements to the end of the array.
<num>是一个非负的数值，与在数组中的位置相一致，基于0的索引。
如果<num>是一个大于等于数组的长度的值，$position修改器没有效果并且表示添加成员的位置为数组末尾。
    { "_id" : 1, "scores" : [ 100 ] }
>db.students.update(
   { _id: 1 },
   {
     $push: {
        scores: {
           $each: [ 50, 60, 70 ],
           $position: 0
        }
     }
   }
)
    { "_id" : 1, "scores" : [  50,  60,  70,  100 ] }
```

4.3.21. $isolated

```
中文隔离的意思.
Prevents a write operation that affects multiple documents from yielding to other reads or writes once
the first document is written. By using the $isolated option, you can ensure that no client sees the
changes until the operation completes or errors out.
防止 一个影响多文档的写入操作 让步于 其他读写

This behavior can significantly affect the concurrency of the system as the operation holds the write
lock much longer than normal.
>db.foo.update(
               { status : "A" , $isolated : 1 },
```

```
               { $inc : { count : 1 } },
               { multi: true }
             )
Without the $isolated operator, the multi-update operation will allow other operations to interleave
with its update of the matched documents.
如果不带$isolated操作符, 多文档更新操作将允许其他操作交替更新自己匹配的文档.
```

4.4. Query Modifiers 查询修改器

```
使用结构:
db.collection.find( { <query> } )._addSpecial(<option> )
db.collection.find( { $query: { <query> }, <option> } )

Modifiers 修改器
Many of these operators have corresponding methods in the shell. These methods provide a straightforward
and user-friendly interface and are the preferred way to add these options.
这些操作符中的很多,在shell中都有对应的方法.这些方法提供了的简单和用户友好的接口,添加这些选项是首选方法。
$comment    Adds a comment to the query to identify queries in the database profiler output.
            添加一个说明用于查询 识别查询在数据库探查器的输出。

$explain    Forces MongoDB to report on query execution plans.
            使MongoDB反馈查询执行的计划。

$hint       Forces MongoDB to use a specific index. See hint()
            强制mongodb使用指定的索引。

$maxScan    Limits the number of documents scanned.
            对扫描文档数量的限制。

$maxTimeMS  Specifies a cumulative time limit in milliseconds for processing operations on a cursor.
                see maxTimeMS().
            指定一个累计时间的限制（毫秒数） 用于一个游标的处理操作。
$max        Specifies an exclusive upper limit for the index to use in a query. See max().
            指定一个专一的上限 用于在查询中的索引
$min        Specifies an inclusive lower limit for the index to use in a query. See min().
            指定一个包含的下限 用于在查询中的索引
$orderby    Returns a cursor with documents sorted according to a sort specification. See sort().
            返回一个游标
$returnKey  Forces the cursor to only return fields included in the index.

$showDiskLoc
            Modifies the documents returned to include references to the  on-disk location of
            each document.

$snapshot   Forces the query to use the index on the _id field.See snapshot().
```

```
$query      Wraps a query document.包装一个查询文档

Sort Order  排序
$natural    A special sort order that orders documents using the order of documents on disk.
```

4.4.1. $comment

```
The $comment meta-operator makes it possible to attach a comment to a query in any context that $query
may appear.
$comment元操作符 使查询附加一个批注成为可能。
Because comments propagate to the profile log, adding a comment can make your profile data easier to
interpret and trace.
因为批注传播到progfile日志中，添加一个评注可以使你的简介数据更容易解释和跟踪。

db.collection.find( { <query> } )._addSpecial( "$comment", <comment> )
db.collection.find( { <query> } ).comment( <comment> )
db.collection.find( { $query: { <query> }, $comment: <comment> } )
```

4.4.2. $explain

```
since version 3.0: Use db.collection.explain() or cursor.explain() instead. The $explain operator
provides information on the query plan. It returns  a document that describes the process and indexes
used to return the query. This may provide useful insight when attempting to optimize a query.

操作符提供查询计划信息,它返回一个描述程序和被用于查询的索引的文档.
当尝试去优化一个查询可以提供有用的感官.
两种使用形式:
db.collection.find()._addSpecial( "$explain", 1 )
db.collection.find( { $query: {}, $explain: 1 } )

在mongo shell中可以通过.explain()检索查询方案。
db.collection.find().explain()
```

4.4.3. $hint

```
The $hint operator forces the query optimizer to use a specific index to fulfill the query.
Specify the index either by the index name or by document
$hint操作符让一个优化器用于一个指定的索引来满足一个查询.
指定索引  可以使用索引名字或者文档
Use $hint for testing query performance and indexing strategies.
    测试查询效率和索引策略.
The mongo shell provides a helper method hint() for the $hint operator.
```

```
>db.users.find().hint( { age: 1 } )
>db.users.find()._addSpecial( "$hint", { age : 1 } )
>db.users.find( { $query: {}, $hint: { age : 1 } } )

>db.users.find( { $query: {}, $hint: { age : 1 }, $explain: 1 } )
```

4.4.4. $maxScan

```
Constrains the query to only scan the specified number of documents when fulfilling the query.
Use this modifier to prevent potentially long running queries from disrupting performance by scanning
through too much data.
约束查询只能扫描指定数量的文档，防止潜在的长时间查询 扫描太多的数据而干扰性能。
Use one of the following forms:

    db.collection.find( { <query> } )._addSpecial( "$maxScan" , <number> )
    db.collection.find( { $query: { <query> }, $maxScan: <number> } )
```

4.4.5. $maxTimeMS

```
The $maxTimeMS operator specifies a cumulative time limit in milliseconds for processing operations on
the cursor. MongoDB interrupts the operation at the earliest following interrupt point.
$maxTimeMS操作符指定一个 ms为单位的计时 限制处理在游标上的操作。
MongoDB在最早的安全中断点中断操作。
db.collection.find().maxTimeMS(100)
db.collection.find( { $query: { }, $maxTimeMS: 100 } )
db.collection.find( { } )._addSpecial("$maxTimeMS", 100)
```

4.4.6. $max

```
http://docs.mongodb.org/manual/reference/operator/meta/max/
Specify a $max value to specify the exclusive upper bound for a specific index in order to constrain
the results of find(). The $max specifies the upper bound for all keys of a specific index in order.
指定一个$max值  指定 特有上界给制定的索引 用以约束find()的结果。

基本使用形式:
db.collection.find( { <query> } )._addSpecial("$max",{ field1: <max value1>, ... fieldN:<max valueN> } )
db.collection.find( { $query: { <query> }, $max: { field1: <max value1>, ...fieldN: <max valueN> } } )

mongo shell:
db.collection.find( { <query> } ).max( { field1: <max value>, ... fieldN: <max valueN> } )

Interaction with Index Selection
交互选择索引
因为max()会请求域上的索引，并强制查询使用这个索引。你可能喜欢$lt操作查询如果可能。
```

```
Because max() requires an index on a field, and forces the query to use this index, you may prefer the
$lt operator for the query if possible. Consider the following example:

db.collection.find( { _id: 7 } ).max( { age: 25 } )
The query uses the index on the age field, even if the index on _id may be better.
这个查询会使用在age域上的索引，即使_id的索引可能会更好。

>db.collection.find( { <query> } ).max( { age: 100 } )
This operation limits the query to those documents where the field age is less than 100 and forces a
query plan which scans the { age: 1 } index from MinKey to 100.
这个操作限制查询 这些文档 当某个文档的age域的值 <100 并且 强制查询方案
扫描{age:1}升序方向的索引从 Minkey 到100.

多索引的选择
{ age: 1, type: -1 }
{ age: 1, type: 1 }
Without explicitly using hint(), MongoDB may select either index for the following operation:
不显式的使用hint(),MongoDB可能选择一个index

>db.collection.find().max( { age: 50, type: 'B' } )

和$min一起使用
限制结果的范围
>db.collection.find().min( { age: 20 } ).max( { age: 25 } )
```

4.4.7. $min

```
使用基本结构:
db.collection.find( { <query> })._addSpecial("$min", {field1:<min value1>,...fieldN: <min valueN> } )
db.collection.find( { $query: {<query> }, $min: { field1: <min value1>, ... fieldN: <min valueN> } } )
mongo shell:
db.collection.find( { <query> } ).min( { field1: <min value>, ... fieldN: <min valueN>} )

假设有一个索引index { age: 1 }
>db.collection.find().min( { age: 20 } )
This operation limits the query to those documents where the field age is at least 20 and forces a
query plan which scans the { age: 1 } index from 20 to MaxKey.
这个操作限制查询  这个文档的 age域至少是20 并且强制一个查询方法 扫描{age:1}升序方向 从20到MaxKey
```

4.4.8. $orderby

```
使用结构:
db.collection.find()._addSpecial( "$orderby", { age : -1 } )
db.collection.find( { $query: {}, $orderby: { age : -1 } } )
The $orderby operator sorts the results of a query in ascending or descending order.
```

```
$orderby操作符对一个查询进行升序(1) 或者降序(-1)的排序。

mongo shell:
db.collection.find().sort( { age: -1 } )
The sort function requires that the entire sort be able to complete within 32 megabytes.
When the sort option consumes more than 32 megabytes, MongoDB will return an error.
sort()函数请求 排序完成需要在32MB字节以内，当消耗超过32MB将会报告一个错误。

To avoid this error, create an index to support the sort operation or use $orderby in conjunction with
$maxScan and/or cursor.limit(). The cursor.limit() increases the speed and reduces the amount of memory
required to return this query by way of an optimized algorithm. The specified limit must result in a
number of documents that fall within the 32 megabyte limit.

为了避免这种错误,创建索引来支持这种操作或使用$orderby为基准进行排序与$maxScan和或cursor.limit()。
cursor.limit()增加的速度和减少所需的内存返回这个查询优化算法。指定的限制必须在32字节的文件数量限制内。
```

4.4.9. $showDiskLoc

```
"$diskLoc": {
            "file": <int>,
            "offset": <int>
            }
使用方法:
    db.collection.find( { <query> } )._addSpecial("$showDiskLoc" , true)
    db.collection.find( { $query: { <query> }, $showDiskLoc: true } )

mongo shell:
    db.collection.find().showDiskLoc()

>db.collection.find( { a: 1 } ).showDiskLoc()
    {
      "_id" : ObjectId("53908ccb18facd50a75bfbac"),
      "a" : 1,
      "b" : 1,
      "$diskLoc" : { "file" : 0, "offset" : 16195760 }
     }
>db.collection.find( { a: 1 }, { $diskLoc: 1 } ).showDiskLoc()
    {
      "_id" : ObjectId("53908ccb18facd50a75bfbac"),
      "$diskLoc" : { "file" : 0, "offset" : 16195760 }
    }
```

4.4.10. $snapshot

```
The $snapshot operator prevents the cursor from returning a document more than once because an
```

```
intervening write operation results in a move of the document.
Even in snapshot mode, objects inserted or deleted during the lifetime of the cursor may or may not be
returned.
$snapshot操作符防止 一个文档返回多次(一个被影响的写操作集合 移动了文档)。

使用方法:
db.collection.find()._addSpecial( "$snapshot", true )
db.collection.find( { $query: {}, $snapshot: true } )

mongo shell:
db.collection.find().snapshot()
```

**You cannot use $snapshot with sharded collections.

4.4.11. $query

```
db.collection.find( { $query: { age : 25 } } )
db.collection.find( { age : 25 } )
```

4.4.12. 注意点

```
###<span style=color:blue>特别注意</span>
$push可以和其他子操作符配合使用
$each:[]
$slice:[-10] $slice的值必须是负值
$sort:{“按哪一个关键字排序”,-1}   -1为降序   1为升序
$push与$slice $sort配合使用的时候  一定要加上$each

需要区分数组和子文档的区别
子文档中还是key-value的形式的文档 {}
数组中的元素仅仅只是value的形式  []
数组中可能有文档  也可能只是值的
```

4.4.13. 备份数据库

```
db.copyDatabase('foobar', 'backdb')
> db.bigdata.count({email:{$
> :/163.com.cn/}})
1220
> db.bigdata.count({email:{$regex:/163.com/}})
278795

###<span style=color:blue>修改文档需注意</span>
```

```
1.db.blog.update({"name":"pyy"},{age:33})
上述语句的本意是修改名字为pyy人的年龄为33
但是 结果是把pyy的所有信息更新为 age = 33
2.查询条件匹配多个文档用于更新
由于第二个参数存在就有产生重复的"_id"的值
数据库会抛异常,导致任何文档都不会更新
3.db.blog.find({key: {$not: 2}});
这个是有问题的，如果只是匹配不等于某个值，应该使用：
db.blog.find({key: {$ne: 2}});

> db.c.update({"comments.name":"t1"},{$set:{"comments.$.size":1}})
> db.getCollectionNames();
    [ "blog", "foobar", "system.indexes", "xunlei" ]
> db.getCollectionNames("foobar");
    [ "blog", "foobar", "system.indexes", "xunlei" ]
> db.getCollectionNames("tutorial");
    [ "blog", "foobar", "system.indexes", "xunlei" ]
> db.blog.find()
    { "_id" : ObjectId("5503ecd83853a67e204962ae"), "title1" : "instrduce of mongo" }
    { "_id" : ObjectId("5503ed873853a67e204962af"), "docment" : "hello mongoDB!" }
    { "_id" : ObjectId("5503edee3853a67e204962b0"), "title" : "mytest", "name" : "pc" }
    { "_id" : ObjectId("5507edd4b63145a5ffc5a59e"), "title" : "book", "comment" : "good" }
    { "_id" : ObjectId("5507f384b63145a5ffc5a59f"), "title" : "book1", "comment" : "bad" }
```