

MongoDB

参考：慕课网 2014 - mongoDB 入门篇(一般般)

参考：黑马 - mongodb 实操视频教程 2017 版

制作日期：2018-8-5

制作人：小桅[yw_forgit@163.com]

第 1 章 数据库基本概念

开源的 NoSQL 数据库。MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的，语法有点类似 javascript 面向对象的查询语言，它是一个面向集合的，模式自由的文档型数据库。

与 MySQL 性能比较

前提：分别插入 100 万条记录，并对其做 100 个用户并发查询操作，。

	插入时间	查询时间
MySQL InnoDB引擎 无索引	10分33秒	39.516秒、35.907秒、39.907秒
MySQL InnoDB引擎 有索引	11分16秒	非常不稳定：22.531秒、13.078秒、23.078秒、26.047秒、21.234秒、28.469秒、20.922秒、13.328秒
MySQL MyISAM引擎 无索引	3分21秒	22.812秒、23.343秒、23.125秒
MySQL MyISAM引擎 有索引	3分50秒	10.312秒、10.359秒、10.296秒
MongoDB 无索引	37秒	59.531秒、60.063秒、59.891秒
MongoDB 有索引	50秒	3.484秒、3.453秒、3.453秒

应用范围和限制。

缺点：不支持连表查询，不支持 sql 语句，不支持事务存储过程等，所以不适合存储数据间关系比较复杂的数据，一般主要是当做一个数据仓库来使用。

适用于：日志系统，股票数据等。。

不用于：电子商务系统等需要连多表查询的功能。

其他



原本这四大天王是这样的



即 LAMP



现在把 MySQL 替换掉



用 MongoDB 的公司有



天天动听
好音质 3亿人的选择



大众点评
dianping.com

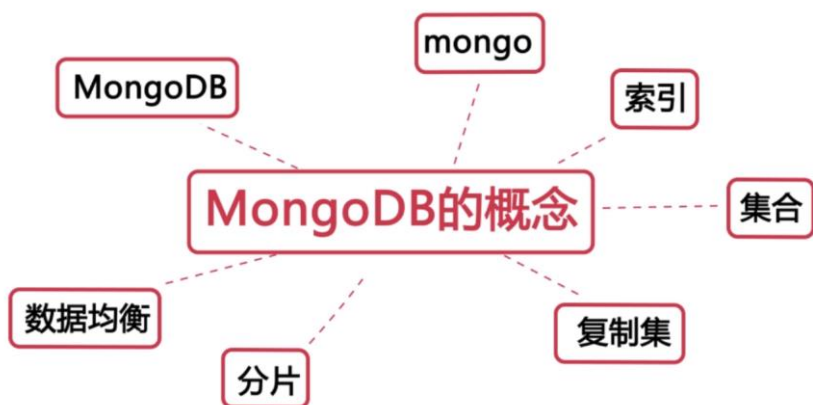


JD.COM 京东



淘宝网
Taobao.com

教程简介



学会MongoDB的搭建

部署数据库服务



熟悉MongoDB的使用

最基本的文档的读写更新删除

各种不同类型的索引的创建与使用

复杂的聚合查询

对数据集进行分片,在不同分片间维持数据均衡

数据备份与恢复

数据迁移

简单运维

- 部署MongoDB集群

- 处理多种常见的故障

- 单节点失效,如何恢复工作

- 数据库意外被杀死如何进行数据恢复

- 数据库发生拒绝服务时如何排查原因

- 数据库磁盘快满时如何处理

课程面向对象

初级 背景知识，基本操作

中级 常见部署操作，简单运维

高级 介绍集群及大型集群的运维经验
及mongoDB的实现原理、常见问题及解决办法

课程面向对象

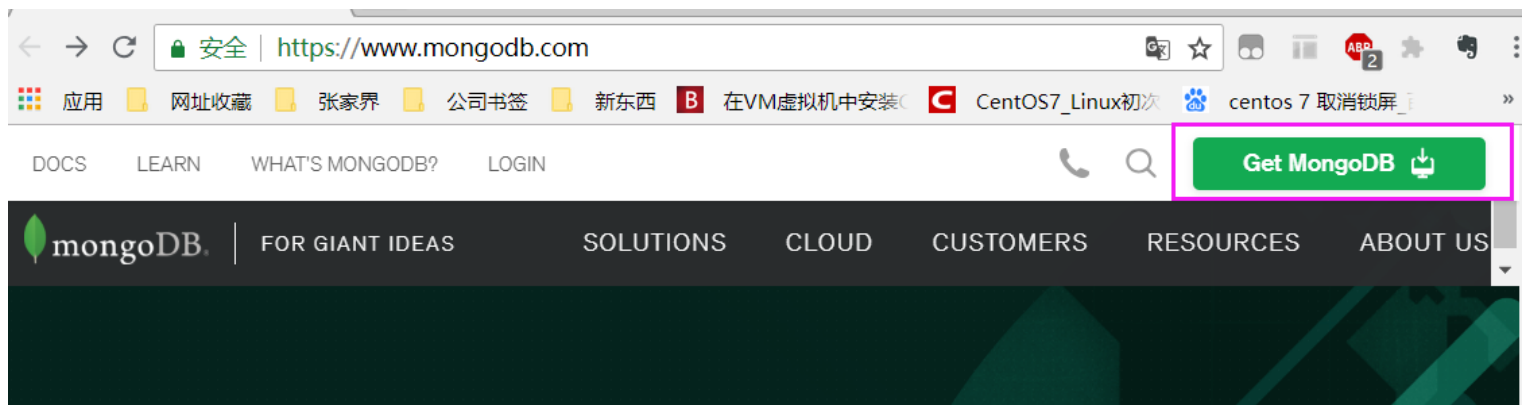
如何运维一个几十T甚至上百T的数据库

如何维持几十个甚至上百个节点的数据库的均衡

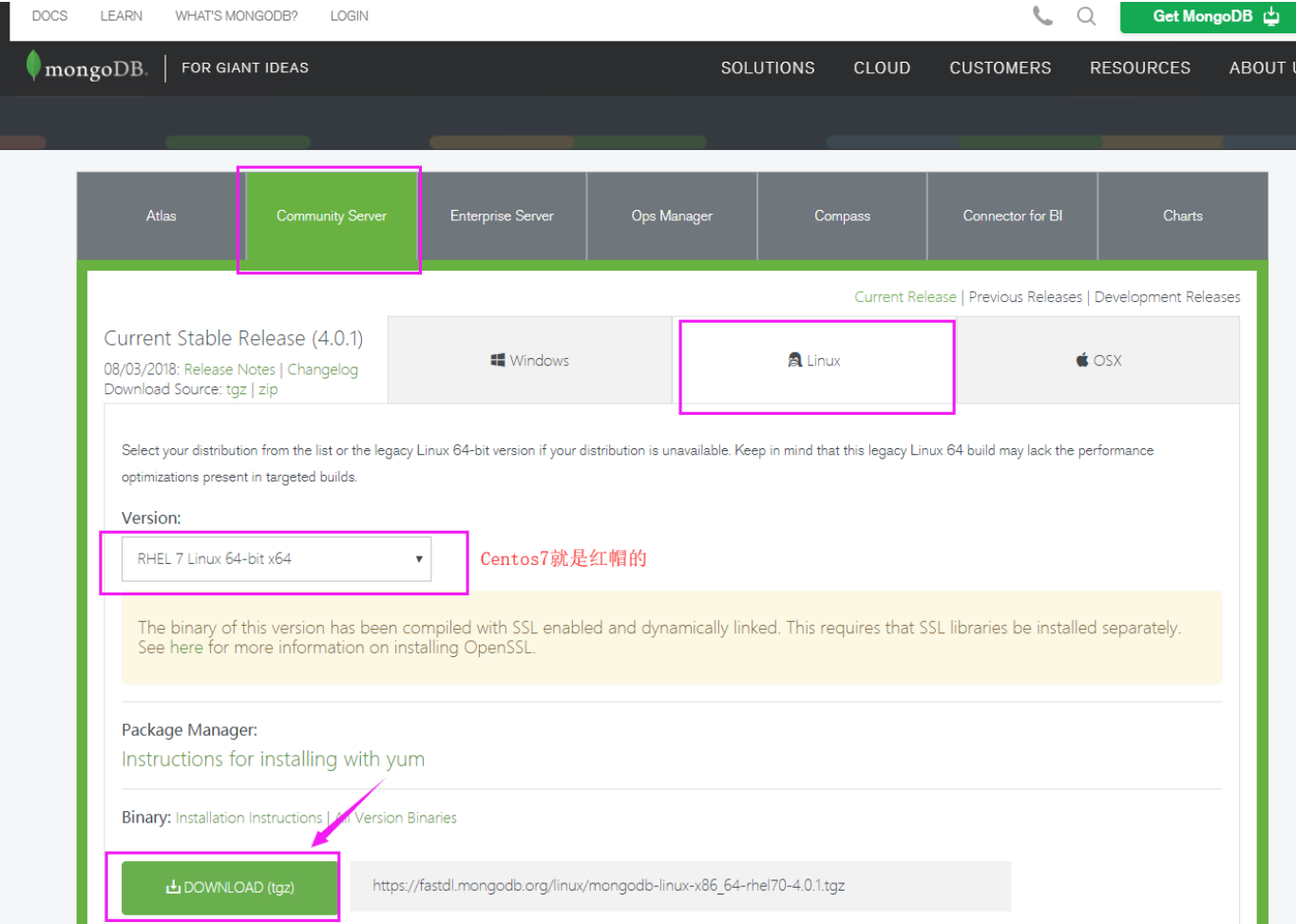
第 2 章 MongoDB 安装与配置

1、Linux 环境

1) 下载、安装



根据自己的系统来下载，我使用 CentOS7



下载完

 mongodb-linux-x86_64-rhel70-4.0.0.tgz	18/8/5 9:33	好压 TGZ 压缩文件	82,146 KB
---	-------------	-------------	-----------

```
// 找个目录上传压缩包到centos
[root@localhost ~]# pwd
/root
// 上传
[root@localhost ~]# rz
[root@localhost ~]# ll
总用量 83764
-rw-----. 1 root root      1968 7月   8 14:36 anaconda-ks.cfg
-rw-r--r--. 1 root root         0 7月  22 00:56 appendonly.aof
-rw-r--r--. 1 root root       93 7月  22 01:12 dump.rdb
-rw-r--r--. 1 root root    2016 7月   8 14:41 initial-setup-ks.cfg
// 就是这个
-rw-r--r--. 1 root root 84116494 8月   5 09:33 mongodb-linux-x86_64-rhel70-4.0.0.tgz
-rw-r--r--. 1 root root      759 7月  22 01:17 nodes-7001.conf
-rw-r--r--. 1 root root      757 7月  22 00:56 nodes-7002.conf
-rw-r--r--. 1 root root      759 7月  22 01:17 nodes-7003.conf
drwxrwxr-x. 6 root root     4096 6月  13 18:47 redis-3.2.12
-rw-r--r--. 1 root root 1551468 7月   8 17:48 redis-3.2.12.tar.gz
-rw-r--r--. 1 root root     73728 7月  18 22:41 redis-3.2.1.gem
drwxr-xr-x. 5 root root        42 7月  22 00:25 redis_cluster
// 解压缩
[root@localhost ~]# tar -zxvf mongodb-linux-x86_64-rhel70-4.0.0.tgz
```

```
// 看看大概
[root@localhost ~]# cd mongodb-linux-x86_64-rhel70-4.0.0/
[root@localhost mongodb-linux-x86_64-rhel70-4.0.0]# ll
总用量 120
drwxr-xr-x. 2 root root 231 8月 5 10:46 bin
-rw-r--r--. 1 root root 34520 6月 22 04:53 GNU-AGPL-3.0
-rw-r--r--. 1 root root 2149 6月 22 04:53 LICENSE-Community.txt
-rw-r--r--. 1 root root 16726 6月 22 04:53 MPL-2
-rw-r--r--. 1 root root 2195 6月 22 04:53 README
-rw-r--r--. 1 root root 57190 6月 22 04:53 THIRD-PARTY-NOTICES
[root@localhost mongodb-linux-x86_64-rhel70-4.0.0]# ls bin

// 启动MongoDB等操作
mongod

// 连接MongoDB
mongo

// 备份
mongodump
mongorestore

// 导入导出
mongoexport
mongoimport

// 查看MongoDB的状态
mongostat

bsondump mongofiles mongoreplay mongos mongotop install_compass
```

2) 配置

搭建简单的 mongodb 服务器

- 1.首先，创建一个叫做 `mongodb_simple` 的目录，进入到目录中。
- 2.创建文件夹：`data`，用来存储数据库的数据文件。
- 3.创建文件夹：`log`，用来存储数据库的日志文件。
- 4.创建文件夹：`bin`，用来存储数据库的可执行文件。
- 5.创建文件夹：`conf`，用来存储数据库的配置文件。

```
[root@localhost mongodb-linux-x86_64-rhel70-4.0.0]#
[root@localhost mongodb-linux-x86_64-rhel70-4.0.0]# cd ..
// 改下名字，太长了
[root@localhost ~]# mv mongodb-linux-x86_64-rhel70-4.0.0 mongodb-linux

[root@localhost ~]# pwd
/root
// 与mongodb-linux同级
[root@localhost ~]# mkdir mongodb_simple
[root@localhost ~]# cd mongodb_simple/
```

```

[root@localhost mongodb_simple]# mkdir data log conf bin
[root@localhost mongodb_simple]# ll
总用量 0
drwxr-xr-x. 2 root root 6 8月 5 10:53 bin
drwxr-xr-x. 2 root root 6 8月 5 10:53 conf
drwxr-xr-x. 2 root root 6 8月 5 10:53 data
drwxr-xr-x. 2 root root 6 8月 5 10:53 log
[root@localhost mongodb_simple]# pwd
/root/mongodb_simple
// 复制到自己创建的bin目录下
[root@localhost mongodb_simple]# cp ../mongodb-linux/bin/mongod bin/
[root@localhost mongodb_simple]# ls bin
mongod
[root@localhost mongodb_simple]# cd conf
[root@localhost conf]# vim mongod.conf
[root@localhost conf]# cat mongod.conf
# 端口
port = 12345
# 数据库保存的路径（相对路径）
dbpath = data
# 日志
logpath = log/mongod.log
# 后台运行，Windows下无效
fork=true

[root@localhost conf]# cd ..
[root@localhost mongodb_simple]# ll bin
总用量 60116
-rwxr-xr-x. 1 root root 61557200 8月 5 10:56 mongod

```

3) 启动

```

// 当前目录的bin, 启动MongoDB, 带配置文件
[root@localhost mongodb_simple]# ./bin/mongod -f conf/mongod.conf
2018-08-05T11:05:36.518+0800 I CONTROL [main] Automatically disabling TLS 1.0, to force-
enable TLS 1.0 specify --sslDisabledProtocols 'none'
about to fork child process, waiting until server is ready for connections.
forked process: 3654
child process started successfully, parent exiting
// 启动成功, 查看目录文件新增了什么
[root@localhost mongodb_simple]# ls data
collection-0-4578585411001061047.wt  index-3-4578585411001061047.wt  mongod.lock
WiredTiger.lock
collection-2-4578585411001061047.wt  index-5-4578585411001061047.wt  sizeStorer.wt
WiredTiger.turtle
collection-4-4578585411001061047.wt  index-6-4578585411001061047.wt  storage.bson
WiredTiger.wt
diagnostic.data                      journal                          WiredTiger
index-1-4578585411001061047.wt      _mdb_catalog.wt                WiredTigerLAS.wt

```



```
[root@localhost mongodb_simple]# ls log
mongod.log
[root@localhost mongodb_simple]#
```

4) 连接

```
[root@localhost mongodb_simple]# ls
bin  conf  data  log
// 复制mongo
[root@localhost mongodb_simple]# cp ../mongodb-linux/bin/mongo bin/
// 查看使用帮助
[root@localhost mongodb_simple]# bin/mongo --help
MongoDB shell version v4.0.0
usage: bin/mongo [options] [db address] [file names (ending in .js)]
db address can be:
    foo                foo database on local machine
    192.168.0.5/foo    foo database on 192.168.0.5 machine
    192.168.0.5:9999/foo  foo database on 192.168.0.5 machine on port 9999
Options:
--shell                run the shell after executing files
--nodb                don't connect to mongod on startup - no
                    'db address' arg expected
--norc                will not run the ".mongorc.js" file on
                    start up
--quiet                be less chatty
--port arg            port to connect to
--host arg            server to connect to
--eval arg            evaluate javascript
-h [ --help ]        show this usage information
--version            show version information
--verbose            increase verbosity
--ipv6                enable IPv6 support (disabled by default)
--disableJavaScriptJIT  disable the Javascript Just In Time
                    compiler
--enableJavaScriptJIT  enable the Javascript Just In Time
                    compiler
--disableJavaScriptProtection  allow automatic JavaScript function
                    marshalling
--ssl                use SSL for all connections
--sslCAFile arg        Certificate Authority file for SSL
--sslPEMKeyFile arg    PEM certificate/key file for SSL
--sslPEMKeyPassword arg    password for key in PEM file for SSL
--sslCRLFile arg        Certificate Revocation List file for SSL
--sslAllowInvalidHostnames    allow connections to servers with
                    non-matching hostnames
--sslAllowInvalidCertificates  allow connections to servers with invalid
                    certificates
--sslFIPSMODE          activate FIPS 140-2 mode at startup
--sslDisabledProtocols arg    Comma separated list of TLS protocols to
                    disable [TLS1_0,TLS1_1,TLS1_2]
```

```
--retryWrites                automatically retry write operations upon
                              transient network errors

--jsHeapLimitMB arg           set the js scope's heap size limit
```

Authentication Options:

```
-u [ --username ] arg         username for authentication
-p [ --password ] arg         password for authentication
--authenticationDatabase arg   user source (defaults to dbname)
--authenticationMechanism arg   authentication mechanism
--gssapiServiceName arg (=mongodb) Service name to use when authenticating
                              using GSSAPI/Kerberos
--gssapiHostName arg           Remote host name to use for purpose of
                              GSSAPI/Kerberos authentication
```

file names: a list of files to run. files have to end in .js **and** will exit after unless --shell is specified

// 使用mongo连接test数据库, test应该是默认有的或者指定test之后自动创建的

```
[root@localhost mongodb_simple]# bin/mongo 127.0.0.1:12345/test
```

MongoDB shell version v4.0.0

connecting to: mongodb://127.0.0.1:12345/test

MongoDB server version: 4.0.0

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>

// 很多警告

Server has startup warnings:

```
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten]
```

```
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten] ** WARNING: Access control is not
enabled for the database.
```

```
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten] **           Read and write access to
data and configuration is unrestricted.
```

```
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten] ** WARNING: You are running this
process as the root user, which is not recommended.
```

```
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten]
```

```
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten] ** WARNING: This server is bound to
localhost.
```

```
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten] **           Remote systems will be
unable to connect to this server.
```

```
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten] **           Start the server with --
bind_ip <address> to specify which IP
```

```
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten] **           addresses it should serve
responses from, or with --bind_ip_all to
```

```
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten] **           bind to all interfaces. If
this behavior is desired, start the
```

```
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten] **           server with --bind_ip
127.0.0.1 to disable this warning.
```

```
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten]
```

```
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten]
```

```
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten] ** WARNING:
```

```
/sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten] **          We suggest setting it to
'never'
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten]
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten] ** WARNING:
/sys/kernel/mm/transparent_hugepage/defrag is 'always'.
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten] **          We suggest setting it to
'never'
2018-08-05T11:05:37.850+0800 I CONTROL [initandlisten]
---
```

Enable MongoDB's free cloud-based monitoring service to collect and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL created **for** you. Anyone you share the URL with will also be able to view **this** page. MongoDB may use **this** information to make product improvements **and** to suggest MongoDB products **and** deployment options to you.

To enable free monitoring, run the following command:

```
db.enableFreeMonitoring()
---
```

```
// 关闭
> db.shutdownServer()
// 提示要用管理员关闭
shutdown command only works with the admin database; try 'use admin'
// 切换管理员
> use admin
switched to db admin
// 关闭
> db.shutdownServer()
server should be down...
2018-08-05T11:24:16.217+0800 I NETWORK [js] trying reconnect to 127.0.0.1:12345 failed
2018-08-05T11:24:16.217+0800 I NETWORK [js] reconnect 127.0.0.1:12345 failed failed
// Ctrl+C强制退出
> ^C
bye
// 看下关闭的日志
[root@localhost mongodb_simple]# tail -f log/mongod.log
2018-08-05T11:21:08.334+0800 I COMMAND [conn1] command admin.$cmd appName: "MongoDB Shell"
command: replSetGetStatus { replSetGetStatus: 1.0, forShell: 1.0, $db: "admin" } numYields:0
ok:0 errMsg:"not running with --replSet" errName:NoReplicationEnabled errCode:76 reslen:122
locks:{} protocol:op_msg 382ms
2018-08-05T11:24:15.718+0800 I COMMAND [conn1] terminating, shutdown command received
{ shutdown: 1.0, $db: "admin" }
2018-08-05T11:24:15.718+0800 I NETWORK [conn1] shutdown: going to close listening
sockets...
2018-08-05T11:24:15.718+0800 I NETWORK [conn1] removing socket file: /tmp/mongodb-
12345.sock
2018-08-05T11:24:15.798+0800 I CONTROL [conn1] Shutting down free monitoring
2018-08-05T11:24:15.798+0800 I FTDC [conn1] Shutting down full-time diagnostic data
capture
```

```
2018-08-05T11:24:15.884+0800 I STORAGE [conn1] WiredTigerKVEngine shutting down
2018-08-05T11:24:16.102+0800 I STORAGE [conn1] shutdown: removing fs lock...
2018-08-05T11:24:16.103+0800 I CONTROL [conn1] now exiting
2018-08-05T11:24:16.103+0800 I CONTROL [conn1] shutting down with code:0
// Ctrl+C强制退出
^C

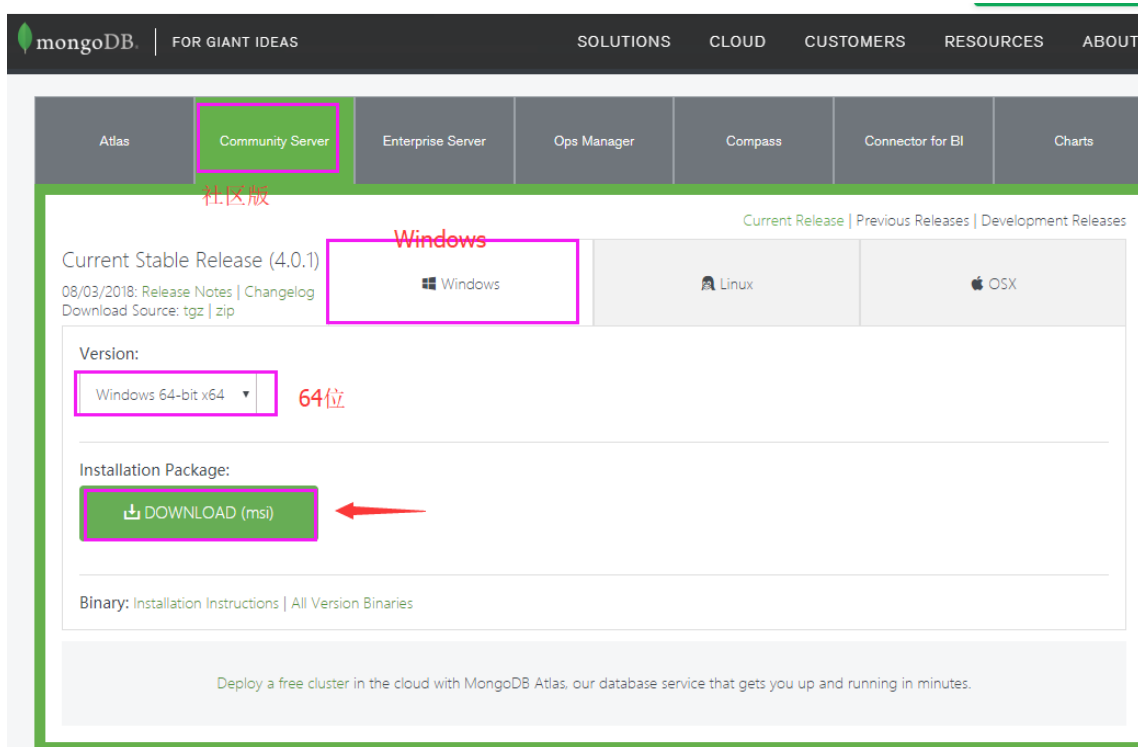
// 重新启动
[root@localhost mongodb_simple]# ./bin/mongod -f conf/mongod.conf
2018-08-05T11:25:30.826+0800 I CONTROL [main] Automatically disabling TLS 1.0, to force-
enable TLS 1.0 specify --sslDisabledProtocols 'none'
about to fork child process, waiting until server is ready for connections.
forked process: 3919
child process started successfully, parent exiting
// 看下启动的日志
[root@localhost mongodb_simple]# tail -f log/mongod.log
2018-08-05T11:25:36.538+0800 I CONTROL [initandlisten]
2018-08-05T11:25:36.539+0800 I CONTROL [initandlisten]
2018-08-05T11:25:36.539+0800 I CONTROL [initandlisten] ** WARNING:
/sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2018-08-05T11:25:36.539+0800 I CONTROL [initandlisten] **          We suggest setting it to
'never'
2018-08-05T11:25:36.539+0800 I CONTROL [initandlisten]
2018-08-05T11:25:36.539+0800 I CONTROL [initandlisten] ** WARNING:
/sys/kernel/mm/transparent_hugepage/defrag is 'always'.
2018-08-05T11:25:36.539+0800 I CONTROL [initandlisten] **          We suggest setting it to
'never'
2018-08-05T11:25:36.539+0800 I CONTROL [initandlisten]
2018-08-05T11:25:36.720+0800 I FTDC [initandlisten] Initializing full-time diagnostic
data capture with directory '/root/mongodb_simple/data/diagnostic.data'
2018-08-05T11:25:37.070+0800 I NETWORK [initandlisten] waiting for connections on port 12345
```

2、Windows 环境

操作数据库的命令是不分平台的。

下载





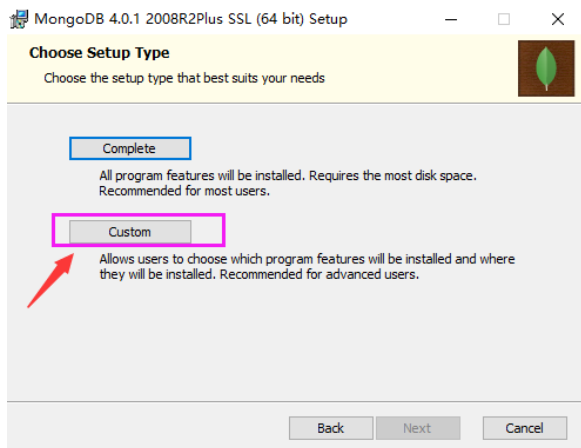
下载完成，是 msi 形式，双击即可打开安装



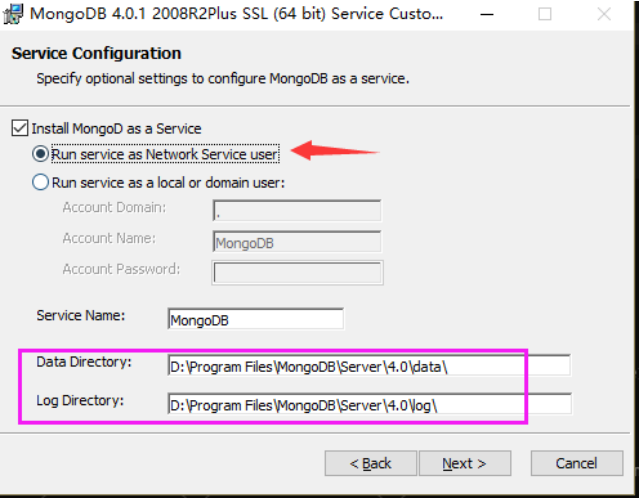
安装

双击 `mongodb-win32-x86_64-2008plus-ssl-4.0.1-signed.msi`，基本都是 Next。

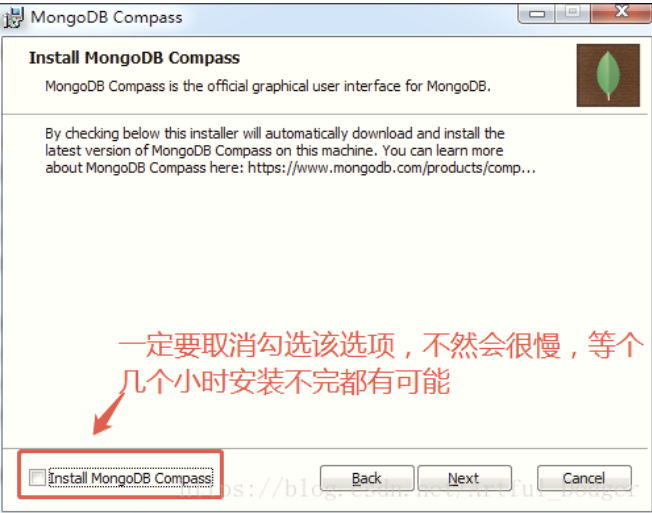
选 自定义安装，主要是为了改安装目录



都是默认的，但是了解下，比如是安装网络服务还是本地，还有数据保存的目录、日志输出的目录。



Compass 是图形化界面，可以不用安装，或者去网上另外找 Compass 的 exe 安装包。但是我的安装还算快的，虽然也有卡住的现象。



安装完成的目录结构，默认下，data 是数据保存目录，log 是日志输出目录。

电脑 > Software (D:) > Program Files > MongoDB > Server > 4.0 >				
名称	修改日期	类型	大小	
bin	18/8/21 11:23	文件夹		
data	18/8/21 11:33	文件夹		
log	18/8/21 11:31	文件夹		
GNU-AGPL-3.0	18/8/3 17:43	0 文件	34 KB	
LICENSE-Community.txt	18/8/3 17:43	TXT 文件	3 KB	
MPL-2	18/8/3 17:43	文件	17 KB	
README	18/8/3 17:43	文件	3 KB	
THIRD-PARTY-NOTICES	18/8/3 17:43	文件	56 KB	

命令认知

电脑 > Software (D:) > Program Files > MongoDB > Server > 4.0 > bin

名称	修改日期	类型	大小
bsondump.exe	18/8/3 17:44	应用程序	8,608 KB
InstallCompass.ps1	18/8/3 18:09	Windows Power...	2 KB
libeay32.dll	18/4/3 18:58	应用程序扩展	2,405 KB
mongo.exe	18/8/3 17:58	应用程序	17,864 KB
mongod.cfg	18/8/21 11:31	CFG 文件	1 KB
mongod.exe	18/8/3 18:11	应用程序	31,434 KB
mongod.pdb	18/8/3 18:11	PDB 文件	343,548 KB
mongodump.exe	18/8/3 17:48	应用程序	10,674 KB
mongoexport.exe	18/8/3 17:46	应用程序	8,862 KB
mongoimport.exe	18/8/3 17:47	应用程序	8,961 KB
mongorestore.exe	18/8/3 17:47	应用程序	11,760 KB
mongos.exe	18/8/3 18:02	应用程序	16,356 KB
mongos.pdb	18/8/3 18:02	PDB 文件	180,108 KB
mongostat.exe	18/8/3 17:45	应用程序	8,950 KB
mongotop.exe	18/8/3 17:49	应用程序	8,758 KB
ssleay32.dll	18/4/3 18:58	应用程序扩展	350 KB

客户端连接程序

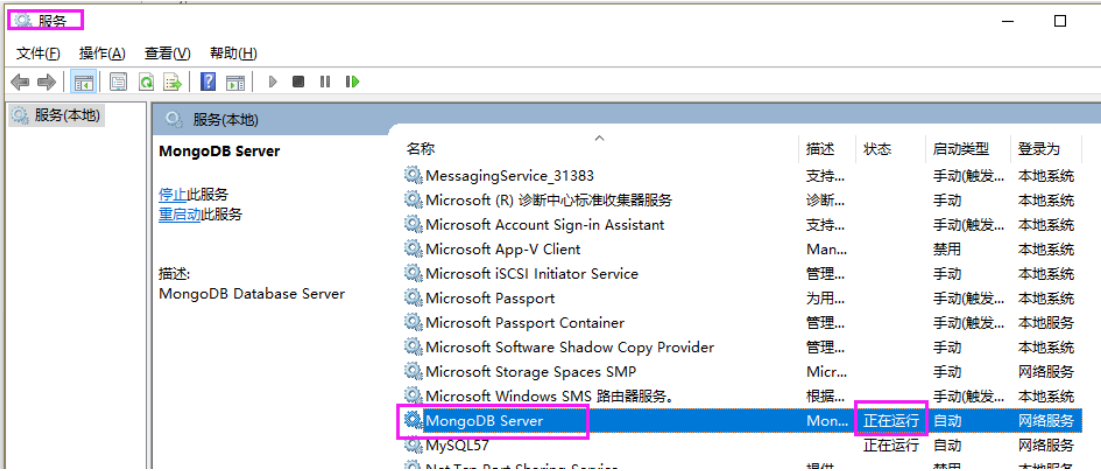
服务器端启动程序

导出数据

导入数据

默认方式启动服务

安装完成，默认是启动的，并且使用默认的数据库目录、日志文件，下图可以看到服务启动了，如果在下图这里手动启动



客户端连接

以 CMD 的方式，进入 MongoDB 的 bin 目录，执行：

mongo ip 地址:端口号 （默认端口号是 27017，如果是本地登录且是默认端口启动服务，则直接输入 mongo 回车即可）

```

D:\Program Files\MongoDB\Server\4.0\bin>mongo
MongoDB shell version v4.0.1
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 4.0.1
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
Server has startup warnings:
2018-08-21T11:31:10.192+0800 I CONTROL [initandlisten]
2018-08-21T11:31:10.192+0800 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-08-21T11:31:10.192+0800 I CONTROL [initandlisten] **           Read and write access to data and configuration is unrestricted.
2018-08-21T11:31:10.192+0800 I CONTROL [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
>

```

自定义启动服务

1) 生产环境（一般用 [Linux 环境](#)）

以管理员的方式运行 CMD，进入 MongoDB 的 bin 目录，执行安装（这是自定义数据库目录、日志文件的安装方式）：

mongod --install --dbpath 数据库存储目录（全路径，带上盘符） **--logpath** 日志文件（全路径，注意不是目录，要定位到文件）

执行完成，同样可以到“[服务](#)”看到 MongoDB 已经启动。[客户端连接](#) 服务也是跟之前一样的

2) 开发环境（直接启动 mongod，无需安装，即无需使用 --install 参数）

以 CMD 的方式，进入 MongoDB 的 bin 目录，执行：

mongod --port --dbpath 数据库存储目录（全路径，带上盘符） **--logpath** 日志文件（全路径，注意不是目录，要定位到文件）

注意：

启动后，该 CMD 窗口不能关闭，一旦关闭，该服务就停止了，不像之前的服务，独立运行于后台。其他操作服务都一样了。

3、帮助命令

1) 未连接服务时：mongod --help

```
D:\Program Files\MongoDB\Server\4.0\bin>mongod --help
```

--logpath // 指定日志文件全路径，带盘符、文件名


```
--dbpath    // 指定数据存放的目录，全路径，带盘符
--auth      // 开启认证
--install   // Windows下，安装服务
--remove    // Windows下，移除服务
--master    // 配置主数据库
--slave     // 配置从数据库
```

2) 连接上服务之后

全局帮助: help

```
> help
    db.help()                help on db methods
    db.mycoll.help()         help on collection methods
    sh.help()                sharding helpers
    rs.help()                replica set helpers
    help admin               administrative help
    help connect             connecting to a db help
    help keys                key shortcuts
    help misc                misc things to know
    help mr                  mapreduce

    show dbs                 show database names
    show collections         show collections in current database
    show users               show users in current database
    show profile             show most recent system.profile entries with time >= 1ms
    show logs                show the accessible logger names
    show log [name]          prints out the last segment of log in memory, 'global' is default
    use <db_name>            set current database
    db.foo.find()            list objects in collection foo
    db.foo.find( { a : 1 } ) list objects in foo where a == 1
    it                       result of the last line evaluated; use to further iterate
    DBQuery.shellBatchSize = x set default number of items to display on shell
    exit                     quit the mongo shell
```

数据库相关帮助: db.help()

```

> db.help()
DB methods:
  db.adminCommand(nameOrDocument) - switches to 'admin' db, and runs command [just calls db.runCommand(...)]
  db.aggregate([pipeline], {options}) - performs a collectionless aggregation on this database; returns a cursor
  db.auth(username, password)
  db.cloneDatabase(fromHost) - deprecated
  db.commandHelp(name) returns the help for the command
  db.copyDatabase(fromdb, todb, fromHost) - deprecated
  db.createCollection(name, {size: ..., capped: ..., max: ...})
  db.createView(name, viewOn, [{operator: ...}], ..., {viewOptions})
  db.createUser(userDocument)
  db.currentOp() displays currently executing operations in the db
  db.dropDatabase()
  db.eval() - deprecated
  db.fsyncLock() flush data to disk and lock server for backups
  db.fsyncUnlock() unlocks server following a db.fsyncLock()
  db.getCollection(cname) same as db['cname'] or db.cname
  db.getCollectionInfos([filter]) - returns a list that contains the names and options of the db's collections
  db.getCollectionNames()
  db.getLastError() - just returns the err msg string
  db.getLastErrorObj() - return full status object
  db.getLogComponents()
  db.getMongo() get the server connection object
  db.getMongo().setSlaveOk() allow queries on a replication slave server
  db.getName()
  db.getPrevError()
  db.getProfilingLevel() - deprecated
  db.getProfilingStatus() - returns if profiling is on and slow threshold
  db.getReplicationInfo()
  db.getSiblingDB(name) get the db at the same server as this one
  db.getWriteConcern() - returns the write concern used for any operations on this db, inherited from server object if set
  db.hostInfo() get details about the server's host
  db.isMaster() check replica primary status
  db.killOp(opid) kills the current operation in the db
  db.listCommands() lists all the db commands
  db.loadServerScripts() loads all the scripts in db.system.js
  db.logout()
  db.printCollectionStats()
  db.printReplicationInfo()
  db.printShardingStatus()
  db.printSlaveReplicationInfo()
  db.dropUser(username)
  db.repairDatabase()
  db.resetError()
  db.runCommand(cmdObj) run a database command. if cmdObj is a string, turns it into {cmdObj: 1}
  db.serverStatus()
  db.setLogLevel(level, <component>)
  db.setProfilingLevel(level, slows) 0=off 1=slow 2=all
  db.setWriteConcern(<write concern doc>) - sets the write concern for writes to the db
  db.unsetWriteConcern(<write concern doc>) - unsets the write concern for writes to the db
  db.setVerboseShell(flag) display extra information in shell output
  db.shutdownServer()
  db.stats()
  db.version() current version of the server

```

集合相关的帮助: db.集合名.help()

```

> db.test_collection.help()
DBCollection help
  db.test_collection.find().help() - show DBCursor help
  db.test_collection.bulkWrite(operations, <optional params>) - bulk execute write operations, optional parameters are: w, wtimeout, j
  db.test_collection.count(query = {}, <optional params>) - count the number of documents that matches the query, optional parameters are: limit, skip, hint, maxTimeMS
  db.test_collection.copyTo(newColl) - duplicates collection by copying all documents to newColl, no indexes are copied.
  db.test_collection.convertToCapped(maxBytes) - calls {convertToCapped: 'test_collection', size:maxBytes} command
  db.test_collection.createIndex(keypattern[, options])
  db.test_collection.createIndexes([keypatterns], <options>)
  db.test_collection.dataSize()
  db.test_collection.deleteOne(filter, <optional params>) - delete first matching document, optional parameters are: w, wtimeout, j
  db.test_collection.deleteMany(filter, <optional params>) - delete all matching documents, optional parameters are: w, wtimeout, j
  db.test_collection.distinct(key, query, <optional params>) - e.g. db.test_collection.distinct('x'), optional parameters are: maxTimeMS
  db.test_collection.drop() drop the collection
  db.test_collection.dropIndex(index) - e.g. db.test_collection.dropIndex("indexName") or db.test_collection.dropIndex({ "indexKey" : 1 })
  db.test_collection.dropIndexes()
  db.test_collection.ensureIndex(keypattern[, options]) - DEPRECATED, use createIndex() instead
  db.test_collection.explain().help() - show explain help
  db.test_collection.reIndex()
  db.test_collection.find([query], [fields]) - query is an optional query filter, fields is optional set of fields to return.
  e.g. db.test_collection.find({x:77}, {name:1, x:1})
  db.test_collection.find(...).count()
  db.test_collection.find(...).limit(n)
  db.test_collection.find(...).skip(n)
  db.test_collection.find(...).sort(...)
  db.test_collection.findOne(query[, [fields], [options], [readConcern]])
  db.test_collection.findOneAndDelete(filter, <optional params>) - delete first matching document, optional parameters are: projection, sort, maxTimeMS
  db.test_collection.findOneAndReplace(filter, replacement, <optional params>) - replace first matching document, optional parameters are: projection, sort, maxTimeMS, upsert, returnNewDocument
  db.test_collection.findOneAndUpdate(filter, update, <optional params>) - update first matching document, optional parameters are: projection, sort, maxTimeMS, upsert, returnNewDocument
  db.test_collection.getDB() get DB object associated with collection
  db.test_collection.getPlanCache() get query plan cache associated with collection
  db.test_collection.getIndexes()
  db.test_collection.group({ key : ..., initial: ..., reduce : ...[, cond: ...] })
  db.test_collection.insert(obj)
  db.test_collection.insertOne(obj, <optional params>) - insert a document, optional parameters are: w, wtimeout, j
  db.test_collection.insertMany([objects], <optional params>) - insert multiple documents, optional parameters are: w, wtimeout, j
  db.test_collection.mapReduce(mapFunction, reduceFunction, <optional params>)
  db.test_collection.aggregate([pipeline], <optional params>) - performs an aggregation on a collection; returns a cursor
  db.test_collection.remove(query)
  db.test_collection.replaceOne(filter, replacement, <optional params>) - replace the first matching document, optional parameters are: upsert, w, wtimeout, j
  db.test_collection.renameCollection(newName, <dropTarget>) renames the collection.
  db.test_collection.runCommand(name, <options>) runs a db command with the given name where the first param is the collection name
  db.test_collection.save(obj)
  db.test_collection.stats({base: N, indexDetails: true/false, indexDetailsKey: <index key>, indexDetailsName: <index name>})
  db.test_collection.storageSize() - includes free space allocated to this collection
  db.test_collection.totalIndexSize() - size in bytes of all the indexes
  db.test_collection.totalSize() - storage allocated for all data and indexes
  db.test_collection.update(query, object[, upsert, bool, multi, bool]) - instead of two flags, you can pass an object with fields: upsert, multi
  db.test_collection.updateOne(filter, update, <optional params>) - update the first matching document, optional parameters are: upsert, w, wtimeout, j
  db.test_collection.updateMany(filter, update, <optional params>) - update all matching documents, optional parameters are: upsert, w, wtimeout, j
  db.test_collection.validate(<full>) - SLOW
  db.test_collection.getShardVersion() - only for use with sharding
  db.test_collection.getShardDistribution() - prints statistics about data distribution in the cluster
  db.test_collection.getSplitKeysForChunks(<maxChunkSize>) - calculates split points over all chunks and returns splitter function
  db.test_collection.getWriteConcern() - returns the write concern used for any operations on this collection, inherited from server/db if set
  db.test_collection.setWriteConcern(<write concern doc>) - sets the write concern for writes to the collection
  db.test_collection.unsetWriteConcern(<write concern doc>) - unsets the write concern for writes to the collection
  db.test_collection.latencyStats() - display operation latency histograms for this collection

```

第 3 章 MongoDB 基本使用

1、插入、查询、更改

```
// 连接MongoDB, 默认使用test数据库了
[root@localhost mongodb_simple]# bin/mongo 127.0.0.1:12345
MongoDB shell version v4.0.0
connecting to: mongodb://127.0.0.1:12345/test
MongoDB server version: 4.0.0
Server has startup warnings:
2018-08-05T11:25:36.538+0800 I CONTROL [initandlisten]
2018-08-05T11:25:36.538+0800 I CONTROL [initandlisten] ** WARNING: Access control is not
enabled for the database.
2018-08-05T11:25:36.538+0800 I CONTROL [initandlisten] **          Read and write access to
data and configuration is unrestricted.
2018-08-05T11:25:36.538+0800 I CONTROL [initandlisten] ** WARNING: You are running this
process as the root user, which is not recommended.
2018-08-05T11:25:36.538+0800 I CONTROL [initandlisten]
2018-08-05T11:25:36.538+0800 I CONTROL [initandlisten] ** WARNING: This server is bound to
localhost.
2018-08-05T11:25:36.538+0800 I CONTROL [initandlisten] **          Remote systems will be
unable to connect to this server.
2018-08-05T11:25:36.538+0800 I CONTROL [initandlisten] **          Start the server with --
bind_ip <address> to specify which IP
2018-08-05T11:25:36.538+0800 I CONTROL [initandlisten] **          addresses it should serve
responses from, or with --bind_ip_all to
2018-08-05T11:25:36.538+0800 I CONTROL [initandlisten] **          bind to all interfaces. If
this behavior is desired, start the
2018-08-05T11:25:36.538+0800 I CONTROL [initandlisten] **          server with --bind_ip
127.0.0.1 to disable this warning.
2018-08-05T11:25:36.538+0800 I CONTROL [initandlisten]
2018-08-05T11:25:36.539+0800 I CONTROL [initandlisten]
2018-08-05T11:25:36.539+0800 I CONTROL [initandlisten] ** WARNING:
/sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2018-08-05T11:25:36.539+0800 I CONTROL [initandlisten] **          We suggest setting it to
'never'
2018-08-05T11:25:36.539+0800 I CONTROL [initandlisten]
2018-08-05T11:25:36.539+0800 I CONTROL [initandlisten] ** WARNING:
/sys/kernel/mm/transparent_hugepage/defrag is 'always'.
2018-08-05T11:25:36.539+0800 I CONTROL [initandlisten] **          We suggest setting it to
'never'
2018-08-05T11:25:36.539+0800 I CONTROL [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service to collect and display
metrics about your deployment (disk utilization, CPU, operation statistics,
etc).

The monitoring data will be available on a MongoDB website with a unique
URL created for you. Anyone you share the URL with will also be able to
```

view [this](#) page. MongoDB may use [this](#) information to make product improvements [and](#) to suggest MongoDB products [and](#) deployment options to you.

To enable free monitoring, run the following command:

```
db.enableFreeMonitoring()
```

```
---
```

```
// 显示所有的数据库
```

```
> show dbs
```

```
admin    0.000GB
```

```
config  0.000GB
```

```
local    0.000GB
```

```
// 切换数据库，如果数据库不存在，则自动创建。如果创建了数据库没有做任何操作，则会自动删除该数据库。
```

```
> use admin
```

```
switched to db admin
```

```
// 切换数据库，没有该数据库会自动创建
```

```
> use simple
```

```
switched to db simple
```

```
// db表示当前数据库，即前面use的数据库；在当前数据库的collection集合插入数据，x=1的数据；collection集合没有创建的话也会自动创建
```

```
> db.collection.insert({x:1})
```

```
WriteResult({ "nInserted" : 1 })
```

```
// 显示所有集合
```

```
> show collections
```

```
collection
```

```
// 查找该集合所有元素，可以看到默认是有_id的，并且_id是唯一的，不可以重复
```

```
> db.collection.find()
```

```
{ "_id" : ObjectId("5b666ee9c0d9e80af34c4840"), "x" : 1 }
```

```
> db.collection.insert({x:2, _id:1})
```

```
WriteResult({ "nInserted" : 1 })
```

```
// 插入重复的_id会报错
```

```
> db.collection.insert({x:3, _id:1})
```

```
WriteResult({
```

```
  "nInserted" : 0,
```

```
  "writeError" : {
```

```
    "code" : 11000,
```

```
    "errmsg" : "E11000 duplicate key error collection: simple.collection index: _id_ dup  
key: { : 1.0 }"
```

```
  }
```

```
})
```

```
> db.collection.find()
```

```
{ "_id" : ObjectId("5b666ee9c0d9e80af34c4840"), "x" : 1 }
```

```
{ "_id" : 1, "x" : 2 }
```

```
// 查找x=1的记录
```

```
> db.collection.find({x:1})
```

```
{ "_id" : ObjectId("5b666ee9c0d9e80af34c4840"), "x" : 1 }
```

```
// 可以for循环插入
> for(i=3; i<7; i++) db.collection.insert({x:i})
WriteResult({ "nInserted" : 1 })
> db.collection.find()
{ "_id" : ObjectId("5b666ee9c0d9e80af34c4840"), "x" : 1 }
{ "_id" : 1, "x" : 2 }
{ "_id" : ObjectId("5b666fb7c0d9e80af34c4841"), "x" : 3 }
{ "_id" : ObjectId("5b666fb7c0d9e80af34c4842"), "x" : 4 }
{ "_id" : ObjectId("5b666fb7c0d9e80af34c4843"), "x" : 5 }
{ "_id" : ObjectId("5b666fb7c0d9e80af34c4844"), "x" : 6 }

// 查询出来的总数
> db.collection.find().count()
6

// 查询结果跳过3条记录，并且只取跳过后的两条记录，排序使用x字段，1表示正向排序，-1表示逆向；对比上面的记录可以看到x=1、2、3的记录被跳过，取x=4、5这两条记录
> db.collection.find().skip(3).limit(2).sort({x:1})
{ "_id" : ObjectId("5b666fb7c0d9e80af34c4842"), "x" : 4 }
{ "_id" : ObjectId("5b666fb7c0d9e80af34c4843"), "x" : 5 }
> db.collection.find({x:1})
{ "_id" : ObjectId("5b666ee9c0d9e80af34c4840"), "x" : 1 }

// 更新x=1为x=99
> db.collection.update({x:1}, {x:99})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

// x=1的记录没有了
> db.collection.find({x:1})

// 变成了x=99
> db.collection.find({x:99})
{ "_id" : ObjectId("5b666ee9c0d9e80af34c4840"), "x" : 99 }

// 一次性插入x y z，这是一条记录
> db.collection.insert({x:100, y:100, z:100})
WriteResult({ "nInserted" : 1 })

// 查询x=100，y、z也会被查询出来，这是一条记录
> db.collection.find({x:100})
{ "_id" : ObjectId("5b667114c0d9e80af34c4845"), "x" : 100, "y" : 100, "z" : 100 }

// 部分更新（是{$set:x:x}）。在z=100的这条记录上，更新y=99，原来y=100的
> db.collection.update({z:100}, {$set:{y:99}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

// 查询z=100，对比一下，发现部分更新y=99了
> db.collection.find({z:100})
{ "_id" : ObjectId("5b667114c0d9e80af34c4845"), "x" : 100, "y" : 99, "z" : 100 }

// 注意，如果不是使用部分更新（{$set:x:x}），结果将完全不一样，z=100这整条记录都会被更新为y=99
```

```

> db.collection.update({z:100}, {y:99})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

// 查询z=100没有结果了, y、z都全部没有了
> db.collection.find({z:100})

// 查询y=99也没有x z了, 这是一个全新的记录y=99
> db.collection.find({y:99})
{ "_id" : ObjectId("5b667114c0d9e80af34c4845"), "y" : 99 }
>

```

修改器:

\$inc // 加一个数字

语法: db.list.update({age:3}, {\$set:{name:'xxx'}})

\$set // 修改某一个字段, 如果该字段不存在就新增该字段

语法: db.list.update({age:3}, {\$inc:{age:10}}) (age 加 10 就是 13 了)

其他:

db.list.find({}, {age:1}) // 1 表示只显示 age 的键值, 如果改为 0 就是不显示 age 键值, 其他都显示

2、增删、删除表和库

```

// 查找没有记录
> db.collection.find({y:100})

// 更新的条件不存在, 不会自动创建
> db.collection.update({y:100}, {y:999})
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
> db.collection.find({y:100})

// 可以看到没有更新到
> db.collection.find({y:999})

// 不满足更新条件, 就直接创建。第三个参数叫multi, 可选, 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.collection.update({y:100}, {y:999}, true)
WriteResult({
  "nMatched" : 0,
  // 这里可留意看到set了一条记录
  "nUpserted" : 1,
  "nModified" : 0,
  "_id" : ObjectId("5b6afaf2b6a03db832921808")
})

// 可看到有记录了
> db.collection.find({y:999})

```

```

{ "_id" : ObjectId("5b6afaf2b6a03db832921808"), "y" : 999 }

// 添加三个c=1
> db.collection.insert({c:1})
WriteResult({ "nInserted" : 1 })
> db.collection.insert({c:1})
WriteResult({ "nInserted" : 1 })
> db.collection.insert({c:1})
WriteResult({ "nInserted" : 1 })

> db.collection.find({c:1})
{ "_id" : ObjectId("5b6afb1711f166a286911d2b"), "c" : 1 }
{ "_id" : ObjectId("5b6afb2511f166a286911d2d"), "c" : 1 }
{ "_id" : ObjectId("5b6afb2711f166a286911d2e"), "c" : 1 }

// 如果有多条更新条件记录存在，则默认只更新第一条
> db.collection.update({c:1}, {c:11})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

// 少了一个c=1
> db.collection.find({c:1})
{ "_id" : ObjectId("5b6afb2511f166a286911d2d"), "c" : 1 }
{ "_id" : ObjectId("5b6afb2711f166a286911d2e"), "c" : 1 }

// 发现只更新了一条
> db.collection.find({c:11})
{ "_id" : ObjectId("5b6afb1711f166a286911d2b"), "c" : 11 }

// >> update的四个参数，本次操作把满足更新条件的记录都更新。
// query : update的查询条件，类似sql update查询内where后面的。
// update : update的对象和一些更新的操作符（如$,$inc...）等，也可以理解为sql update查询内set后面的。
upsert : 可选，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 : 可选，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。
// writeConcern : 可选，抛出异常的级别。
> db.collection.update({c:1}, {$set:{c:11}}, false, true)
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })

// 可以看到全部都更新了
> db.collection.find({c:11})
{ "_id" : ObjectId("5b6afb1711f166a286911d2b"), "c" : 11 }
{ "_id" : ObjectId("5b6afb2511f166a286911d2d"), "c" : 11 }
{ "_id" : ObjectId("5b6afb2711f166a286911d2e"), "c" : 11 }

// 删除必须要带条件的
> db.collection.remove()
2018-08-08T22:18:02.572+0800 E QUERY [js] Error: remove needs a query :
DBCollection.prototype._parseRemove@src/mongo/shell/collection.js:356:1
DBCollection.prototype.remove@src/mongo/shell/collection.js:383:18
@(shell):1:1
> db.collection.remove({c:11})

```

```
WriteResult({ "nRemoved" : 3 })
```

```
// 删除表
```

```
> db.collection.drop()
```

```
true
```

```
// 可以看到没有表、集合了
```

```
> show tables
```

```
> show collections
```

```
// 删除当前数据库
```

```
> db.dropDatabase()
```

常用操作符：

\$lt // 小于

\$lte // 小于等于

\$gt // 大于

\$gte // 大于等于

\$ne // 不等于

\$in

\$nin

\$or // 或

\$not

\$mod // 取模

\$exists // 是否存在

\$where

3、操作表的基本概念

1) 文档

文档是 mongoDB 中数据的**基本单元**，类似关系数据库的**行**，多个键值对有序地放置在一起便是文档。

MongoDB 中以文档的方式存取记录，如几条记录格式如下：

```
{username:"Tom", "age":10, sex:"男"} // age是数字类型
```

```
{username:"Tom", "age":"10"} // age是字符串类型
```

```
{Username:"Tom"} // Username 是大写开头
```

注意：

(1)以上是几个不同的文档，MongoDB **区分大小写**的数据类型

(2)每一个文档尺寸不能超过 16M

(1) 文档就是键值对，数据类型是 **BSON** 格式，支持的值更加丰富。

BSON是JSON的扩展他先新增了诸如日期,浮点等JSON不支持的数据类型

BSON	
null	用于表示空或者不存在的字段
布尔	两个个数值true和false
32位何64位整数	Shell中不支持 需用到其他高级语言的驱动来完成,JS不可使用.
64位浮点	Shell中使用的数字其实是这种类型(x:3.414)
UTF-8	其实就是字符串类型
对象ID	内置默认ID对象({_id:ObjectId()})
日期	{x:new Date()}
正则	{x:/uspcat/i}
Javascript代码块	{x:function(){...}}
undefined	为定义类型注意他和null不是一个类型
数组	{gps:[20,56]}
内嵌文档	{x:{name:"uspcat"}}
二进制	任意字节的字符串shell中时无法使用的

ObjectId 类型:

每个文档都有一个_id 字段，并且同一个结合的_id 值是唯一的。

ObjectId 对象数据组成: 时间戳|机器码|PID|计数器。(PID 是线程 id)

_id 的键值也可以自定义输入，但是不能重复

2) 集合

集合就是一组文档，**多个文档组成一个集合**，集合类似于 mysql 里面的**表**。

无模式是指，在同一个集合中可以包含不同格式的文档，如:

```
{username:"Tom", "age":"10"}    // age是字符串类型
{Username:"Tom"}    // Username 是大写开头
```

以上两个文档可以放在同一个集合中。在 **Mysql** 需要先建表再插入数据。

模式自由(schema-free): 意思是集合里面**没有行和列的概念**。

注意:

MongoDB 中的**集合不用创建、没有结构**，所以可以放不同格式的文档。

3) 数据库

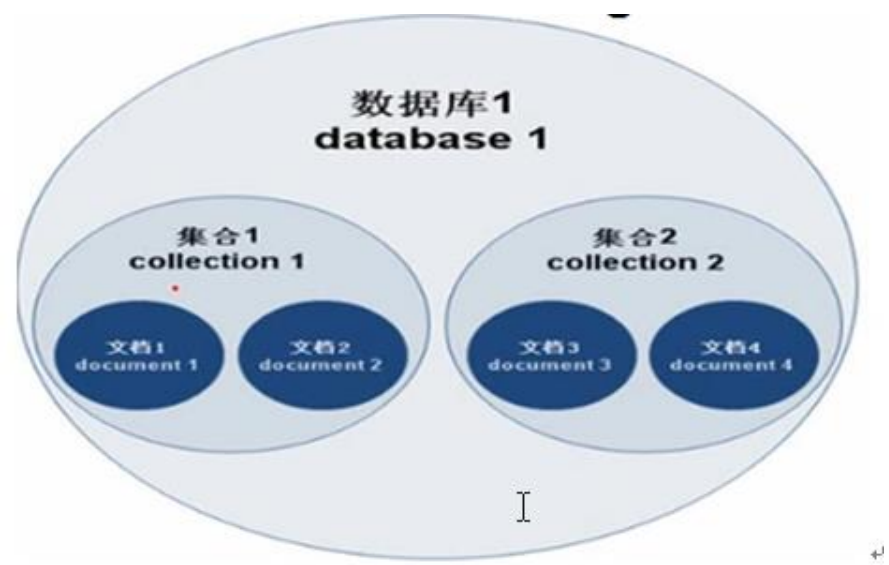
多个集合可以组成数据库。一个 mongoDB 实例可以承载多个数据库，他们之间完全独立。Mongodb 中的数据库和 Mysql 中的数据库概念类似，只是**无需创建**。

一个数据库中可以有多个集合。

一个集合中可以有多个文档。

4) MongoDB 的数据体系

Mongodb	mysql
文档(document) (单个文档最大16M)	记录(row)
集合(collection)	表(table)
数据库(database) (32位系统上，一个数据库的文件大小不能超过2G)	数据库(database)



第 4 章 mongodb 常见的查询索引

索引的种类

- 1._id 索引
- 2.单键索引
- 3.多键索引
- 4.复合索引
- 5.过期索引
- 6.全文索引
- 7.地理位置索引

准备工作

```
// 之前创建的，有bin、conf的目录
[root@localhost ~]# cd mongodb_simple/
[root@localhost mongodb_simple]# ll
总用量 4
```

```

drwxr-xr-x. 2 root root  33 8月  5 11:19 bin
drwxr-xr-x. 2 root root  25 8月  5 10:58 conf
drwxr-xr-x. 4 root root 4096 8月  7 21:40 data
drwxr-xr-x. 2 root root 100 8月  7 21:38 log
[root@localhost mongodb_simple]# ls bin/
mongo mongod
[root@localhost mongodb_simple]# ls conf/
mongod.conf
// 连接
[root@localhost mongodb_simple]# bin/mongod -f conf/mongod.conf
2018-08-08T21:43:42.081+0800 I CONTROL [main] Automatically disabling TLS 1.0, to force-
enable TLS 1.0 specify --sslDisabledProtocols 'none'
about to fork child process, waiting until server is ready for connections.
forked process: 2692
child process started successfully, parent exiting
[root@localhost mongodb_simple]# bin/mongo 127.0.0.1:12345
MongoDB shell version v4.0.0
connecting to: mongodb://127.0.0.1:12345/test
MongoDB server version: 4.0.0
Server has startup warnings:
。。。。。省略

// 显示数据库
> show dbs
admin  0.000GB
config 0.000GB
local  0.000GB
simple  0.000GB

// 使用simple数据库
> use simple
switched to db simple

// 显示表或者集合
> show tables

```

1、_id 索引

_id 索引，是绝大多数集合默认建立的索引。

对于每个插入的数据，MongoDB 都会自动生成一条唯一的_id 字段。

```

// 插入记录，并且自动创建t_index表
> db.t_index.insert({x:1})
WriteResult({ "nInserted" : 1 })
> db.t_index.find()
{ "_id" : ObjectId("5b6af43611f166a286911d1e"), "x" : 1 }

// 查看索引
> db.t_index.getIndexes()

```

```
[
  {
    "v" : 2,
    // 可以看到默认_id就是建立的索引
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "simple.t_index"
  }
]

> db.t_index.insert({x:1, y:2, z:3})
WriteResult({ "nInserted" : 1 })
> db.t_index.find()
{ "_id" : ObjectId("5b6af43611f166a286911d1e"), "x" : 1 }
{ "_id" : ObjectId("5b6af49611f166a286911d1f"), "x" : 1, "y" : 2, "z" : 3 }
>
```

2、单键索引

- 1) 单键索引是最普通的索引
- 2) 与_id索引不同,单键索引不会自动创建

例如:一条记录,形式为:{x:1,y:2,z:3}

```
// 建立单键索引,索引字段是x,其中1表示正向索引,-1表示逆向索引,后面只要涉及索引的创建,都是这个意思,并不是x=1的意思
> db.t_index.ensureIndex({x:1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
> db.t_index.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "simple.t_index"
  },
  {
    "v" : 2,
    // 可以看到x字段被建立索引
    "key" : {
      "x" : 1
    },

```

```

        "name" : "x_1",
        "ns" : "simple.t_index"
    }
]

// 查找所有，索引不起作用的
> db.t_index.find()
{ "_id" : ObjectId("5b6af43611f166a286911d1e"), "x" : 1 }
{ "_id" : ObjectId("5b6af49611f166a286911d1f"), "x" : 1, "y" : 2, "z" : 3 }

// 要查找单独的x索引，才起作用
> db.t_index.find({x:1})
{ "_id" : ObjectId("5b6af43611f166a286911d1e"), "x" : 1 }
{ "_id" : ObjectId("5b6af49611f166a286911d1f"), "x" : 1, "y" : 2, "z" : 3 }

```

3、多键索引

多键索引与单键索引创建形式相同，区别在于字段的值。

单键索引：值为一个单一的值,例如字符串、数字或者日期。

多键索引：值具有多个记录，例如数组。

```

// x字段之前已经建立索引
> db.t_index.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "simple.t_index"
  },
  {
    "v" : 2,
    "key" : {
      "x" : 1
    },
    "name" : "x_1",
    "ns" : "simple.t_index"
  }
]

// x是多值的时候，此时就是多键索引了
> db.t_index.insert({x:[1,2,3,4]})
WriteResult({ "nInserted" : 1 })
> db.t_index.getIndexes()
[
  {
    "v" : 2,

```

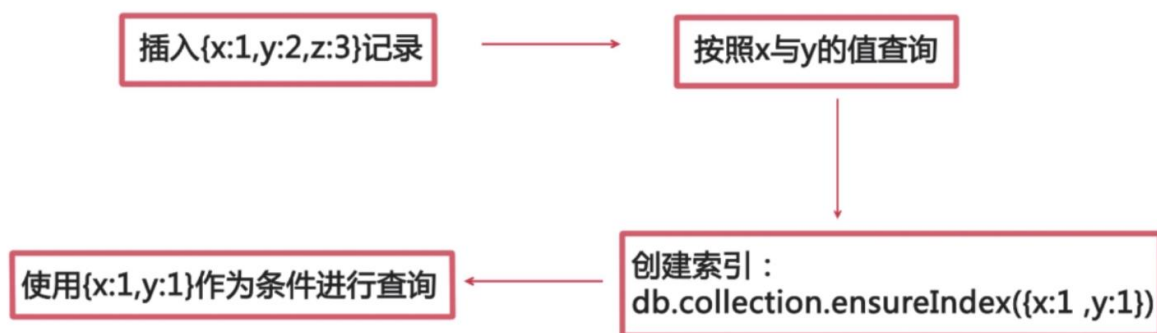
```

    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "simple.t_index"
  },
  {
    "v" : 2,
    "key" : {
      "x" : 1
    },
    "name" : "x_1",
    "ns" : "simple.t_index"
  }
]

```

4、复合索引

当我们的查询条件不只有一个时，就需要建立复合索引



```

// 建立复合索引，1表示x、y都是正向索引
> db.t_index.ensureIndex({x:1, y:1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 2,
  "numIndexesAfter" : 3,
  "ok" : 1
}
> db.t_index.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "simple.t_index"
  }
]

```

```

    },
    {
      "v" : 2,
      "key" : {
        "x" : 1
      },
      "name" : "x_1",
      "ns" : "simple.t_index"
    },
    {
      "v" : 2,
      "key" : {
        "x" : 1,
        "y" : 1
      },
      // 复合索引
      "name" : "x_1_y_1",
      "ns" : "simple.t_index"
    }
  ]

// 要复合查询才起作用
> db.t_index.find({x:1, y:2})
{ "_id" : ObjectId("5b6af49611f166a286911d1f"), "x" : 1, "y" : 2, "z" : 3 }

```

5、过期索引

- 1) 过期索引:是在一段时间后会过期的索引。
- 2) 在索引过期后,相应的数据会被删除。
- 3) 这适合存储一些在一段时间之后会失效的数据比如用户的登录信息、存储的日志。
- 4) 建立方法:

```
db.collection.ensureIndex({time:1}, {expireAfterSeconds:10})
```

```

// 创建过期索引, time:1表示正向索引, expireAfterSeconds是过期时间, 单位是秒
> db.t_index.ensureIndex({time:1}, {expireAfterSeconds:30})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 3,
  "numIndexesAfter" : 4,
  "ok" : 1
}

// 必须插入一个时间
> db.t_index.insert({time:new Date()})
WriteResult({ "nInserted" : 1 })

// 现在是插入time=1的值, 比较一下
> db.t_index.insert({time:1})
WriteResult({ "nInserted" : 1 })

```

```

> db.t_index.find()
{ "_id" : ObjectId("5b6af43611f166a286911d1e"), "x" : 1 }
{ "_id" : ObjectId("5b6af49611f166a286911d1f"), "x" : 1, "y" : 2, "z" : 3 }
{ "_id" : ObjectId("5b6af56111f166a286911d20"), "x" : [ 1, 2, 3, 4 ] }
{ "_id" : ObjectId("5b6af62411f166a286911d21"), "time" : ISODate("2018-08-08T13:54:44.093Z") }
{ "_id" : ObjectId("5b6af62d11f166a286911d22"), "time" : 11 }

// 过了几分钟之后再查询
> db.t_index.find()
{ "_id" : ObjectId("5b6af43611f166a286911d1e"), "x" : 1 }
{ "_id" : ObjectId("5b6af49611f166a286911d1f"), "x" : 1, "y" : 2, "z" : 3 }
{ "_id" : ObjectId("5b6af56111f166a286911d20"), "x" : [ 1, 2, 3, 4 ] }
// 发现带时间的那个time没有了, 过期就会被删除, 而time=11由于不符合时间格式, 故不会生效
{ "_id" : ObjectId("5b6af62d11f166a286911d22"), "time" : 11 }

```

过期索引的限制

- 1) 存储在过期索引字段的值必须是指定的时间类型。

说明: 必须是 `ISODate` 或者 `ISODate` 数组, 不能使用时间戳, 否则不能被自动删除。

- 2) 如果指定了 `ISODate` 数组, 则按照最小的时间进行删除。

- 3) 过期索引不能是复合索引。

- 4) 删除时间不是精确。

说明: 删除过程是由后台程序每 60s 跑一次, 而且删除也需要一些时间, 所以存在误差。

6、子文档索引

语法: ↵

`db.集合名.ensureIndex({filed.subfield:1/-1});` ↵

如下文档可以建立子文档索引 ↵

`{name:'诺基亚手机 1',price:12.34,spc:{weight:100,area:'纽约'}} ↵`

`{name:'诺基亚手机 2',price:42.34,spc:{weight:200,area:'伦敦'}} ↵`

比如要查询 `weight` 等于 100 的文档。 ↵

`db.goods.find({'spc.weight':100})` ↵ **weight就是子文档**

↵

根据当前案例, 我们建立子文档索引 ↵

`db.net.ensureIndex({'spc.w':1})` ↵ **这里是weight**

7、重建索引

一个表经过很多次修改后,导致表的文件产生空洞,索引文件也如此. ↵

可以通过索引的重建,减少索引文件碎片,并提高索引的效率. ↵

类似 mysql 中的 `optimize table` ↵

mysql 里面使用 `optimize table` 语法: `optimize table` 表名 ↵

`db.集合名.reIndex()` ↵

↵

第 5 章 全文索引(重点)

对字符串与字符串数组创建全文可搜索的索引

适用情况::

`{author:"", titile:"", article:""}`

建立方法:

`db.articles.ensureIndex(key:"text")`

`db.articles.ensureIndex(key1:" text", key2:' text")`

`db.articles.ensureIndex("$**":"text")`

集合中所有字段都创建索引

```
// 建立全文索引, 即给article建立全文索引, 使用text
> db.t_index.ensureIndex({"article":"text"})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 4,
  "numIndexesAfter" : 5,
  "ok" : 1
}
> db.t_index.getIndexes ()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "simple.t_index"
  },
  {
    "v" : 2,
    "key" : {
      "x" : 1
    },
    "name" : "x_1",
    "ns" : "simple.t_index"
```

```

},
{
  "v" : 2,
  "key" : {
    "x" : 1,
    "y" : 1
  },
  "name" : "x_1_y_1",
  "ns" : "simple.t_index"
},
{
  "v" : 2,
  "key" : {
    "time" : 1
  },
  "name" : "time_1",
  "ns" : "simple.t_index",
  "expireAfterSeconds" : 30
},
{
  "v" : 2,
  // 这里
  "key" : {
    "_fts" : "text",
    "_ftsx" : 1
  },
  // 这里
  "name" : "article_text",
  "ns" : "simple.t_index",
  "weights" : {
    "article" : 1
  },
  "default_language" : "english",
  "language_override" : "language",
  "textIndexVersion" : 3
}
]

// 插入数据
> db.t_index.insert({"article":"aa bb cc dd"})
WriteResult({ "nInserted" : 1 })
> db.t_index.insert({"article":"aa bb"})
WriteResult({ "nInserted" : 1 })
> db.t_index.insert({"article":"aa bb cc"})
WriteResult({ "nInserted" : 1 })
> db.t_index.insert({"article":"aa bb rr dd"})
WriteResult({ "nInserted" : 1 })

```

如何使用全文索引查询

```

db.articles.find( { $text: { $search: "coffee" } })
db.articles.find( { $text: { $search: "aa bb cc" } })

```

```
db.articles.find( { $text: { $search: "aa bb -cc" }})
db.articles.find( { $text: { $search: "\"aa\" bb cc" }})
```

```
// 使用全文索引查询
> db.t_index.find({$text:{$search:"aa"}})
// 全部匹配
{ "_id" : ObjectId("5b6af7111f166a286911d24"), "article" : "aa bb" }
{ "_id" : ObjectId("5b6af71a11f166a286911d25"), "article" : "aa bb cc" }
{ "_id" : ObjectId("5b6af72811f166a286911d26"), "article" : "aa bb rr dd" }
{ "_id" : ObjectId("5b6af70a11f166a286911d23"), "article" : "aa bb cc dd" }

// 只有一个rr
> db.t_index.find({$text:{$search:"rr"}})
{ "_id" : ObjectId("5b6af72811f166a286911d26"), "article" : "aa bb rr dd" }

// aa bb cc是或的关系，只要一个满足即可
> db.t_index.find({$text:{$search:"aa bb cc"}})
{ "_id" : ObjectId("5b6af7111f166a286911d24"), "article" : "aa bb" }
{ "_id" : ObjectId("5b6af71a11f166a286911d25"), "article" : "aa bb cc" }
{ "_id" : ObjectId("5b6af72811f166a286911d26"), "article" : "aa bb rr dd" }
{ "_id" : ObjectId("5b6af70a11f166a286911d23"), "article" : "aa bb cc dd" }

// -cc是不等于cc的
> db.t_index.find({$text:{$search:"aa bb -cc"}})
{ "_id" : ObjectId("5b6af7111f166a286911d24"), "article" : "aa bb" }
{ "_id" : ObjectId("5b6af72811f166a286911d26"), "article" : "aa bb rr dd" }

// 对aa bb cc都加双引号，表示与的关系，注意双引号要转移
> db.t_index.find({$text:{$search:"\"aa\" \"bb\" \"cc\""}})
{ "_id" : ObjectId("5b6af71a11f166a286911d25"), "article" : "aa bb cc" }
{ "_id" : ObjectId("5b6af70a11f166a286911d23"), "article" : "aa bb cc dd" }
```

全文索引相似度：

\$meta 操作符：{ score: { \$meta: "textScore" } }

写在查询条件后面可以返回结果的相似度。

与 sort 一起使用，可以达到很好的实用效果。

```
// 全文索引相似度查询，score就是相似度
> db.t_index.find({$text:{$search:"aa bb"}}, {score:{$meta:"textScore"}})
// 前面两个最低
{ "_id" : ObjectId("5b6af70a11f166a286911d23"), "article" : "aa bb cc dd", "score" : 1.25 }
{ "_id" : ObjectId("5b6af72811f166a286911d26"), "article" : "aa bb rr dd", "score" : 1.25 }
{ "_id" : ObjectId("5b6af71a11f166a286911d25"), "article" : "aa bb cc", "score" :
1.3333333333333333 }
// 最高
{ "_id" : ObjectId("5b6af7111f166a286911d24"), "article" : "aa bb", "score" : 1.5 }

// 对相似度排序
> db.t_index.find({$text:{$search:"aa bb"}},
{score:{$meta:"textScore"}}).sort({score:{$meta:"textScore"}})
```

```

{ "_id" : ObjectId("5b6af7111f166a286911d24"), "article" : "aa bb", "score" : 1.5 }
{ "_id" : ObjectId("5b6af71a11f166a286911d25"), "article" : "aa bb cc", "score" :
1.3333333333333333 }
{ "_id" : ObjectId("5b6af70a11f166a286911d23"), "article" : "aa bb cc dd", "score" : 1.25 }
{ "_id" : ObjectId("5b6af72811f166a286911d26"), "article" : "aa bb rr dd", "score" : 1.25 }

// 现在是4.0版本, 已经支持中文的全文索引了, 但是注意查询的关键字要有空格隔开, 英文的也是
> db.t_index.insert({"article":"中文bb"})
WriteResult({"nInserted" : 1 })
> db.t_index.insert({"article":"中文 存储"})
WriteResult({"nInserted" : 1 })
> db.t_index.find({$text:{$search:"中文"}})
// 有空格的只有一个
{ "_id" : ObjectId("5b6af8a411f166a286911d28"), "article" : "中文 存储" }

// 英文的也要空格隔开
> db.t_index.insert({"article":"aabb"})
WriteResult({"nInserted" : 1 })
> db.t_index.find({$text:{$search:"aa"}})
{ "_id" : ObjectId("5b6af7111f166a286911d24"), "article" : "aa bb" }
{ "_id" : ObjectId("5b6af71a11f166a286911d25"), "article" : "aa bb cc" }
{ "_id" : ObjectId("5b6af72811f166a286911d26"), "article" : "aa bb rr dd" }
{ "_id" : ObjectId("5b6af70a11f166a286911d23"), "article" : "aa bb cc dd" }

```

全文索引使用限制

全文索引非常强大但是同样存在限制:

每次查询,只能指定一个\$text 查询

\$text 查询不能出现在\$nor 查询中

查询中如果包含了\$text, hint 不再起作用

很可惜,MongoDB 全文索引还不支持中文(2014 年), 2018 年已经支持

第 6 章 索引属性(重点)

创建索引时的格式:

db.collection.ensureIndex({param}, {param}) 其中,第二个参数便是索引的属性

重要的属性:

1) 名字, name指定:

```
db.collection.ensureIndex({}, {name:""})
```

```
// 创建复合索引
```

```
> db.properties.ensureIndex({x:1, y:1})
{
  "createdCollectionAutomatically" : true,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```

```
// 查看索引
```

```
> db.properties.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "simple.properties"
  },
  {
    "v" : 2,
    "key" : {
      "x" : 1,
      "y" : 1
    },
    // 可以看到索引默认的名字，很无聊
    "name" : "x_1_y_1",
    "ns" : "simple.properties"
  }
]
```

```
// 指定索引名称进行创建索引，使用name
```

```
> db.properties.ensureIndex({x:1, y:1, z:1}, {name:"normal_index"})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 2,
  "numIndexesAfter" : 3,
  "ok" : 1
}
> db.properties.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "simple.properties"
  },
  {
    "v" : 2,
    "key" : {
      "x" : 1,
      "y" : 1,
      "z" : 1
    },
    "name" : "normal_index",
    "ns" : "simple.properties"
  }
]
```

```

{
  "v" : 2,
  "key" : {
    "x" : 1,
    "y" : 1
  },
  "name" : "x_1_y_1",
  "ns" : "simple.properties"
},
{
  "v" : 2,
  "key" : {
    "x" : 1,
    "y" : 1,
    "z" : 1
  },
  // 这样索引就一目了然了
  "name" : "normal_index",
  "ns" : "simple.properties"
}
]

// 删除索引还可以直接使用索引名称
> db.properties.dropIndex("normal_index")
{ "nIndexesWas" : 3, "ok" : 1 }
> db.properties.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "simple.properties"
  },
  {
    "v" : 2,
    "key" : {
      "x" : 1,
      "y" : 1
    },
    "name" : "x_1_y_1",
    "ns" : "simple.properties"
  }
]
>

```

2) 唯一性, unique指定:

```
db.collection.ensureIndex({}, {unique:true/false})
```

唯一性就是该字段（建立了唯一索引）的值不能重复，包括空也不能重复。

```
// 建立唯一索引
> db.tUnique.ensureIndex({m:1, n:1}, {unique:true})
{
  "createdCollectionAutomatically" : true,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
// 插入第一条
> db.tUnique.insert({m:1, n:2})
WriteResult({ "nInserted" : 1 })
// 重复插入，直接报错了
> db.tUnique.insert({m:1, n:2})
WriteResult({
  "nInserted" : 0,
  "writeError" : {
    "code" : 11000,
    "errmsg" : "E11000 duplicate key error collection: simple.tUnique index: m_1_n_1 dup
key: { : 1.0, : 2.0 }"
  }
})
>
```

3) 稀疏性，sparse指定：

```
db.collection.ensureIndex({}, {sparse:true/false})
```

eg:

创建唯一索引，指定稀疏性：

```
db.good.ensureIndex({"goodId": 1}, {"unique": true, "sparse": true});
```

我们知道，唯一索引只允许一条索引字段为空的记录存在，之后就不允许插入了。再次插入 goodId 为 null（goodId 不是等于 1 哈，1 是正向索引）的记录时会报错：E11000 duplicate key error index: dup key: { : null };

“sparse=true”的作用就是当 goodId 在文档中不存在，或为空值，则不进入索引，从而避免上述问题。

4) 是否定时删除，expireAfterSeconds指定（TTL，过期索引）：

```
db.collection.ensureIndex({time:ISODate}, {expireAfterSeconds:秒数})
```

之前讲过

第 7 章 地理位置索引(重点)

概念:将一些点的位置存储在MongoDB中,创建索引后,可以按照位置来查找其他点。

分类:

2d索引,用于存储和查找平面上的点.

2dsphere索引,用于存储和查找球面上的点.

查找方式:

1. 查找距离某个点一定距离内的点.

2. 查找包含在某区域内的点.

1、2d 索引

2D索引:平面地理位置索引

2Dsphere索引:球面地理位置索引

创建方式:

```
db.collection.ensureIndex({w:"2d" }) // w 有没有双引号都可以
```

位置表示方式:经纬度[经度,纬度]

取值范围:经度[-180,180] 纬度[-90,90]

查询方式:

(1)\$near 查询:

查询距离某个点最近的点.默认查询100条最近的

```
// 创建2D索引
> db.location_test.ensureIndex({w:"2d"})
{
  "createdCollectionAutomatically" : true,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}

// 插入值, [经度值, 维度值]
> db.location_test.insert({w:[1, 1]})
WriteResult({ "nInserted" : 1 })
> db.location_test.insert({w:[1, 2]})
WriteResult({ "nInserted" : 1 })
> db.location_test.insert({w:[3, 2]})
WriteResult({ "nInserted" : 1 })
> db.location_test.insert({w:[100, 100]})
WriteResult({ "nInserted" : 1 })
```



```
// 经度超过范围了
> db.location_test.insert({w:[200, 100]})
WriteResult({
  "nInserted" : 0,
  "writeError" : {
    "code" : 13027,
    "errmsg" : "point not in interval of [ -180, 180 ] :: caused by :: { _id:
ObjectId('5b6e79391cda0f0d675baf93'), w: [ 200.0, 100.0 ] }"
  }
})
> db.location_test.insert({w:[180, 100]})
WriteResult({ "nInserted" : 1 })

// 查询距离[1,1]这个点的数据，默认查询100个最接近的
> db.location_test.find({w:{$near:[1,1]}})
{ "_id" : ObjectId("5b6e79151cda0f0d675baf8f"), "w" : [ 1, 1 ] }
{ "_id" : ObjectId("5b6e791c1cda0f0d675baf90"), "w" : [ 1, 2 ] }
{ "_id" : ObjectId("5b6e79211cda0f0d675baf91"), "w" : [ 3, 2 ] }
{ "_id" : ObjectId("5b6e79321cda0f0d675baf92"), "w" : [ 100, 100 ] }
{ "_id" : ObjectId("5b6e79481cda0f0d675baf94"), "w" : [ 180, 100 ] }

// maxDistance是要求距离该点的距离小于等于10
> db.location_test.find({w:{$near:[1,1], $maxDistance:10}})
{ "_id" : ObjectId("5b6e79151cda0f0d675baf8f"), "w" : [ 1, 1 ] }
{ "_id" : ObjectId("5b6e791c1cda0f0d675baf90"), "w" : [ 1, 2 ] }
{ "_id" : ObjectId("5b6e79211cda0f0d675baf91"), "w" : [ 3, 2 ] }

// minDistance要求最小距离是3
> db.location_test.find({w:{$near:[1,1], $maxDistance:10, $minDistance:3}})
> db.location_test.find({w:{$near:[1,1], $maxDistance:10, $minDistance:1}})
{ "_id" : ObjectId("5b6e791c1cda0f0d675baf90"), "w" : [ 1, 2 ] }
{ "_id" : ObjectId("5b6e79211cda0f0d675baf91"), "w" : [ 3, 2 ] }
```

(2) \$geoWithin 查询:

查询某个形状内的点。

形状表示:

1. \$box: 矩形, 使用

{ \$box: [[<x1>, <y1>], [<x2>, <y2>]] } 表示. // 坐标是两个对角点, 这两个点分别向x、y轴做高, 就得到了矩形

2. \$center: 圆形使用

{ \$center: [[<x1>, <y1>], r] } 表示. // 圆心点、半径

3. \$polygon: 多边形, 使用

{ \$polygon: [[<x1>, <y1>], [<x2>, <y2>], [<x3>, <y3>]] } 表示. // 三个点就是三角形, 以此类推

```
// 查询在该两点所围成的矩形内的记录
> db.location_test.find({w:{$geoWithin:{$box:[[0,0], [3,3]]}}})
{ "_id" : ObjectId("5b6e79151cda0f0d675baf8f"), "w" : [ 1, 1 ] }
```

```

{ "_id" : ObjectId("5b6e791c1cda0f0d675baf90"), "w" : [ 1, 2 ] }
{ "_id" : ObjectId("5b6e79211cda0f0d675baf91"), "w" : [ 3, 2 ] }
> db.location_test.find({w:{$geoWithin:{$box:[[0,0], [2,3]]}}})
{ "_id" : ObjectId("5b6e79151cda0f0d675baf8f"), "w" : [ 1, 1 ] }
{ "_id" : ObjectId("5b6e791c1cda0f0d675baf90"), "w" : [ 1, 2 ] }

// 查询在该圆内的记录
> db.location_test.find({w:{$geoWithin:{$center:[[0,0], 5]]}}})
{ "_id" : ObjectId("5b6e79151cda0f0d675baf8f"), "w" : [ 1, 1 ] }
{ "_id" : ObjectId("5b6e791c1cda0f0d675baf90"), "w" : [ 1, 2 ] }
{ "_id" : ObjectId("5b6e79211cda0f0d675baf91"), "w" : [ 3, 2 ] }

// 查询在该多边形内的记录
> db.location_test.find({w:{$geoWithin:{$polygon:[[0,0],[0,1],[2,5],[6,1]]}}})
{ "_id" : ObjectId("5b6e79151cda0f0d675baf8f"), "w" : [ 1, 1 ] }
{ "_id" : ObjectId("5b6e791c1cda0f0d675baf90"), "w" : [ 1, 2 ] }
{ "_id" : ObjectId("5b6e79211cda0f0d675baf91"), "w" : [ 3, 2 ] }
>

```

(3)geoNear 查询:

geoNear使用runCommand命令进行使用,会返回比较详细的信息,常用使用如下:

```

db.runCommand(
  {geoNear:<collection>,    // 查询哪个集合
  near:[x,y],    // 距离哪个点附近
  minDistance:xxx,    // 最小距离
  maxDistance:xxx,    // 最大距离
  num:xxx,    // 查询多少条记录
  ...    // 还可以添加其他查询条件
  })

```

```

// 集合写错了,会报下面这个错误
> db.runCommand({geoNear:"localtion_test", near:[1,1], minDistance:1, maxDistance:10,
num:2})
{ "ok" : 0, "errmsg" : "can't find ns" }

// 查询
> db.runCommand({geoNear:"location_test", near:[1,1], minDistance:1, maxDistance:10, num:2})
{
  // 返回结果有两条记录
  "results" : [
    {
      // 距离[1,1]这个点的距离
      "dis" : 1,
      "obj" : {
        "_id" : ObjectId("5b6e791c1cda0f0d675baf90"),
        "w" : [
          1,
          2
        ]
      }
    }
  ]
}

```

```
    }
  },
  {
    // 距离[1,1]这个点的距离
    "dis" : 2.23606797749979,
    "obj" : {
      "_id" : ObjectId("5b6e79211cda0f0d675baf91"),
      "w" : [
        3,
        2
      ]
    }
  },
  ],
  "stats" : {
    "nscanned" : 29, // 扫描的数
    "objectsLoaded" : 3,
    "avgDistance" : 1.618033988749895, // 返回结果中所有点距离查询点([1,1])平均距离
    "maxDistance" : 2.23606797749979, // 最大距离的
    "time" : 1209 // 查询时间
  },
  // 1=查询成功
  "ok" : 1
}
>
```

2、2dsphere 索引

概念:球面地理位置索引

创建方式:

```
db.collection.ensureIndex({w:" 2dsphere" })
```

位置表示方式:

GeoJSON:描述一个点，一条直线，多边形等形状。

格式:

```
{type:"", coordinates:[ <coordinates> ]}
```

查询方式与 2d 索引查询方式类似

第 8 章 索引构建情况分析

索引好处:加快索引相关的查询。

索引不好处:增加磁盘空间消耗,降低写入性能。

如何评判当前索引构建情况：

1、mongostat 工具介绍.

mongostat:查看mongodb运行状态的程序.

使用说明: `mongostat -h 127.0.0.1:12345`

字段说明:

索引情况: `idx miss`

```
[root@localhost ~]# ll
总用量 83764
-rw-----. 1 root root      1968 7月  8 14:36 anaconda-ks.cfg
-rw-r--r--. 1 root root         0 7月 22 00:56 appendonly.aof
-rw-r--r--. 1 root root       93 7月 22 01:12 dump.rdb
-rw-r--r--. 1 root root     2016 7月  8 14:41 initial-setup-ks.cfg
drwxr-xr-x. 3 root root      120 8月  5 10:46 mongodb-linux
-rw-r--r--. 1 root root 84116494 8月  5 09:33 mongodb-linux-x86_64-rhel70-4.0.0.tgz
drwxr-xr-x. 6 root root       52 8月  5 10:53 mongodb_simple
-rw-r--r--. 1 root root      759 7月 22 01:17 nodes-7001.conf
-rw-r--r--. 1 root root      757 7月 22 00:56 nodes-7002.conf
-rw-r--r--. 1 root root      759 7月 22 01:17 nodes-7003.conf
drwxrwxr-x. 6 root root     4096 6月 13 18:47 redis-3.2.12
-rw-r--r--. 1 root root 1551468 7月  8 17:48 redis-3.2.12.tar.gz
-rw-r--r--. 1 root root    73728 7月 18 22:41 redis-3.2.1.gem
drwxr-xr-x. 5 root root       42 7月 22 00:25 redis_cluster

// 这是MongoDB的解压目录, 有些bin没有复制到mongodb_simple, 故来到解压的目录执行
[root@localhost ~]# cd mongodb-linux/
[root@localhost mongodb-linux]# ll
总用量 120
drwxr-xr-x. 2 root root    231 8月  5 10:46 bin
-rw-r--r--. 1 root root 34520 6月 22 04:53 GNU-AGPL-3.0
-rw-r--r--. 1 root root  2149 6月 22 04:53 LICENSE-Community.txt
-rw-r--r--. 1 root root 16726 6月 22 04:53 MPL-2
-rw-r--r--. 1 root root  2195 6月 22 04:53 README
-rw-r--r--. 1 root root 57190 6月 22 04:53 THIRD-PARTY-NOTICES
[root@localhost mongodb-linux]# ls bin
bsondump          mongo  mongodump  mongofiles  mongoreplay  mongos  mongotop
install_compass  mongod  mongoexport  mongoimport  mongorestore  mongostat

// 执行bin下的mongostat, 可以查看mongodb的运行状态
[root@localhost mongodb-linux]# ./bin/mongostat -h 127.0.0.1:12345
insert query update delete getmore command dirty used flushes vsize  res grw arw net_in
net_out conn          time
 *0 *0 *0 *0 0 3|0 0.0% 0.0% 0 1.03G 46.0M 0|0 1|0 269b 103k
2 Aug 11 14:42:52.061
 *0 *0 *0 *0 0 2|0 0.0% 0.0% 0 1.03G 46.0M 0|0 1|0 158b 60.7k
2 Aug 11 14:42:53.059
 *0 *0 *0 *0 0 1|0 0.0% 0.0% 0 1.03G 46.0M 0|0 1|0 157b 60.4k
2 Aug 11 14:42:54.062
 *0 *0 *0 *0 0 2|0 0.0% 0.0% 0 1.03G 46.0M 0|0 1|0 158b 61.0k
```

```

2 Aug 11 14:42:55.056
....

// 会一直监测，想退出按Ctrl+C，强制退出
[root@localhost mongodb-linux]#

// 在连接上的会话中操作循环插入数据
> for(i=0;i<10000; i++) db.status_test.insert({x:i})
// 这条结果不会立马出现，等执行完就有了 WriteResult({ "nInserted" : 1 })
>

// 这边立马再监测，看看
[root@localhost mongodb-linux]# ./bin/mongostat -h 127.0.0.1:12345
insert query update delete getmore command dirty used flushes vsize res qrw arw net_in
net_out conn time
 3171    *0    *0    *0      0    2|0  0.8% 0.8%      0 1.03G 48.0M 0|0 1|0   393k   204k
2 Aug 11 14:44:27.563
 3042    *0    *0    *0      0    1|0  1.0% 1.0%      0 1.03G 49.0M 0|0 1|0   377k   198k
2 Aug 11 14:44:28.563
 3001    *0    *0    *0      0    1|0  1.2% 1.3%      0 1.03G 50.0M 0|0 1|0   372k   196k
2 Aug 11 14:44:29.564
 3051    *0    *0    *0      0    2|0  1.5% 1.5%      0 1.03G 50.0M 0|0 1|0   379k   198k
2 Aug 11 14:44:30.562
 3041    *0    *0    *0      0    2|0  1.7% 1.7%      0 1.03G 50.0M 0|0 1|0   377k   198k
2 Aug 11 14:44:31.560
...

```

2、profile 集合介绍.

开发环境使用profile可以看到是否设置合理，但是到了线上，不要开启，会影响性能，因为一直记录操作的记录

3、日志介绍.

设置日志输出级别

在配置文件配置: verbose = vvvvv // 五个级别

4、explain 分析.

explain 可以看到本次查询是否使用了索引、查询时间等信息。（4.0 版本好像看不到查询时间）

```

> db.exp.insert({x:1})
WriteResult({ "nInserted" : 1 })
> db.exp.find({x:1}).explain()
{
  "queryPlanner" : {
    "plannerVersion" : 1,

```

```

    "namespace" : "simple.exp",
    "indexFilterSet" : false,
    "parsedQuery" : {
      "x" : {
        "$eq" : 1
      }
    },
    "winningPlan" : {
      "stage" : "COLLSCAN",
      "filter" : {
        "x" : {
          "$eq" : 1
        }
      },
      "direction" : "forward"
    },
    "rejectedPlans" : [ ]
  },
  "serverInfo" : {
    "host" : "localhost.centos",
    "port" : 12345,
    "version" : "4.0.0",
    "gitVersion" : "3b07af3d4f471ae89e8186d33bbb1d5259597d51"
  },
  "ok" : 1
}
> db.exp.ensureIndex({x:1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
> db.exp.find({x:1}).explain()
{
  "queryPlanner" : {
    "plannerVersion" : 1,
    "namespace" : "simple.exp",
    "indexFilterSet" : false,
    "parsedQuery" : {
      "x" : {
        "$eq" : 1
      }
    },
    "winningPlan" : {
      "stage" : "FETCH",
      "inputStage" : {
        "stage" : "IXSCAN",
        "keyPattern" : {
          "x" : 1
        },
        "indexName" : "x_1",

```

```

        "isMultiKey" : false,
        "multiKeyPaths" : {
          "x" : [ ]
        },
        "isUnique" : false,
        "isSparse" : false,
        "isPartial" : false,
        "indexVersion" : 2,
        "direction" : "forward",
        "indexBounds" : {
          "x" : [
            "[1.0, 1.0]"
          ]
        }
      },
      "rejectedPlans" : [ ]
    },
    "serverInfo" : {
      "host" : "localhost.centos",
      "port" : 12345,
      "version" : "4.0.0",
      "gitVersion" : "3b07af3d4f471ae89e8186d33bbb1d5259597d51"
    },
    "ok" : 1
  }
>

```

粗略看查询时间，复制下面代码，直接粘贴到命令行上，即可输出查询时间。

```

var start = new Date()
db.java.find({name:'thinking'})
var end = new Date()
end - start

```

第 9 章 导出导入

导出、导入的操作，可以是本地的 MongoDB 服务器，也可以是远程的，而命令参数也是通用的。

```

-h // host主机
--port // 端口
-d // 指明使用的库
-c // 指明要导出的集合
-o // 指明要导出的文件名
--csv // 指定导出的csv格式
-q // 过滤导出
-f // field1 field2 列名

```

-u // 用户名
-p // 密码

注意：如果使用用户名、密码，则要使用超级管理员的，如果端口是默认的，可以不指定端口

1) 导出数据

先往集合，插入一条记录

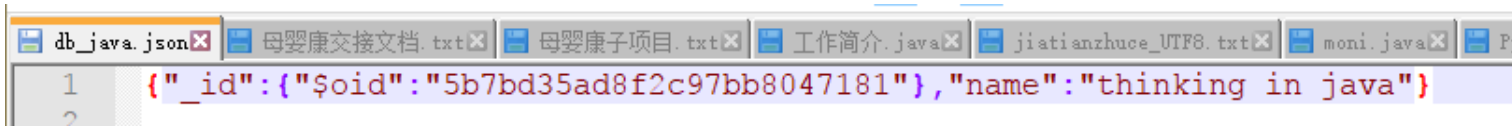
```
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
> use test
switched to db test
> db.java.insert({name:'thinking in java'})
WriteResult({"nInserted" : 1})
> db.java.find()
{"_id" : ObjectId("5b7bd35ad8f2c97bb8047181"), "name" : "thinking in java" }
```

mongoexport 导出命令

mongoexport -h localhost --port 27017 -d test -c java -o e:/db_java.json

```
D:\Program Files\MongoDB\Server\4.0\bin>mongoexport -h localhost --port 27017 -d test -c java -o e:/db_java.json
2018-08-21T16:56:37.176+0800 connected to: localhost:27017
2018-08-21T16:56:37.199+0800 exported 1 record
```

打开导出的文件



2) 导入数据

-d // 待导入的数据库
-c // 待导入的集合，不存在会自动创建
--type // csv或json，默认是json
--file // 导入的文件路径

mongoimport 命令，导入

mongoimport -h localhost --port 27017 -d test_import -c java --file e:/db_java.json

```
D:\Program Files\MongoDB\Server\4.0\bin>mongoimport -h localhost --port 27017 -d test_import -c java --file e:/db_java.json
2018-08-21T17:11:02.643+0800 connected to: localhost:27017
2018-08-21T17:11:02.832+0800 imported 1 document
D:\Program Files\MongoDB\Server\4.0\bin>
```

然后查询一下

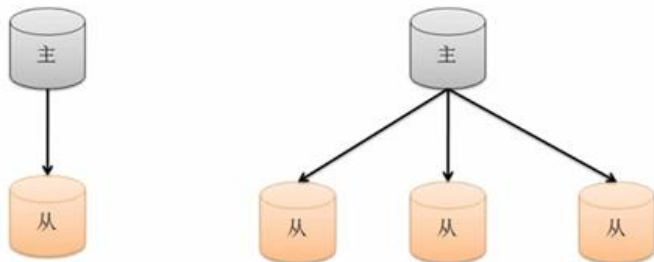

```
> use test_import
switched to db test_import
> db.java.find()
{ "_id" : ObjectId("5b7bd35ad8f2c97bb8047181"), "name" : "thinking in java" }
```

第 10 章 主从复制（读写分离）

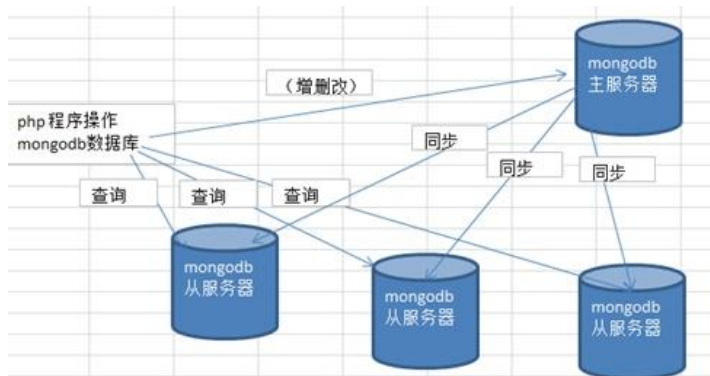
注意：4.0 不支持，所以后面都是完成不了的。

主从复制是一个简单的数据库同步备份的集群技术。

至少两台数据库服务器，可以分别设置主服务器和从服务器，对主服务器的任何操作都会同步到从服务器上。



1、原理图

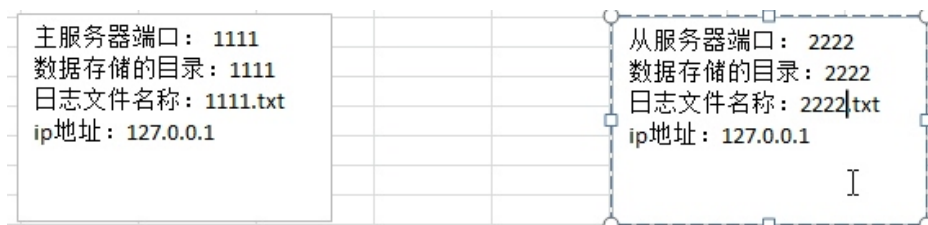


2、实现的注意点

- 1) 在数据库集群中要明确的知道谁是主服务器，主服务器只有一台。
- 2) 从服务器要知道自己的数据源是谁，也就是自己属于哪个主服务器。
- 3) `--master` 用来确定主服务器，`--slave` 和 `--source` 来控制从服务器

3、配置步骤

可以通过不同的端口来模拟多台 MongoDB 服务器。



0) 创建数据库目录、日志文件

此电脑 > Data (E:) > SoftwareData > MongoDB >

名称	修改日期	类型	大小
1111	18/8/21 17:23	文件夹	
2222	18/8/21 17:24	文件夹	
data	18/8/21 17:14	文件夹	
1111.txt	18/8/21 17:24	TXT 文件	0 KB
2222.txt	18/8/21 17:24	TXT 文件	0 KB
log.txt	18/8/21 16:53	TXT 文件	0 KB
log.txt.2018-08-21T07-49-52	18/8/21 15:48	2018-08-21T07-...	0 KB
log.txt.2018-08-21T08-53-17	18/8/21 16:53	2018-08-21T08-...	11 KB

1) 启动主服务器

`mongod --port 1111 --dbpath E:\SoftwareData\MongoDB\1111\ --logpath E:\SoftwareData\MongoDB\1111.txt --master`
 what, 居然说不再支持主从复制了。

```

D:\Program Files\MongoDB\Server\4.0\bin>mongod --port 1111 --dbpath E:\SoftwareData\MongoDB\1111\ --logpath E:\SoftwareData\MongoDB\1111.txt --master
2018-08-21T17:40:15.585+0800 F CONTROL [main] Master/slave replication is no longer supported
D:\Program Files\MongoDB\Server\4.0\bin>
  
```

查看帮助 `mongod --help`, 也是这么说, 就不管了吧。。。。。。。。。。但是网上搜不到 Master/slave replication no longer supported 的结果。

```

--master
--slave

Master/slave replication no longer supported
Master/slave replication no longer supported
  
```

2) 启动从服务器

本来如果可以的话:

(1) 启动主服务器

```

C:\uamp\nongodb\bin>mongod --port 1111 --logpath c:\uamp\nongodb\1111.txt --dbpath c:\uamp\nongodb\1111 --master
all output going to: c:\uamp\nongodb\1111.txt
  
```

(2) 启动从服务器

```

C:\uamp\nongodb\bin>mongod --port 2222 --dbpath c:\uamp\nongodb\2222 --logpath c:\uamp\nongodb\2222.txt --slave --source
127.0.0.1:1111
all output going to: c:\uamp\nongodb\2222.txt
  
```

(3) 客户端登陆到主服务器

添加一些数据, 测试是否同步到从服务器

如下在主服务器里面, 添加了一些文档

第一步: 客户端登录到主服务器, 添加一些文档

```

C:\uamp\nongodb\bin>mongo localhost:1111
MongoDB shell version: 2.0.3
connecting to: localhost:1111/test
> use stu
switched to db stu
> db.php.insert({name:'xiaobai',age:12})
> db.php.insert({name:'xiaohei',age:22})
> db.php.find()
{ "_id" : ObjectId("58ef37e09d2cac9e181bfc1b"), "name" : "xiaobai", "age" : 12 }
{ "_id" : ObjectId("58ef37f09d2cac9e181bfc1c"), "name" : "xiaohei", "age" : 22 }
  
```

第二步: 登陆到从服务器, 查看是否有数据

第二步：登陆到从服务器，查看是否有数据。

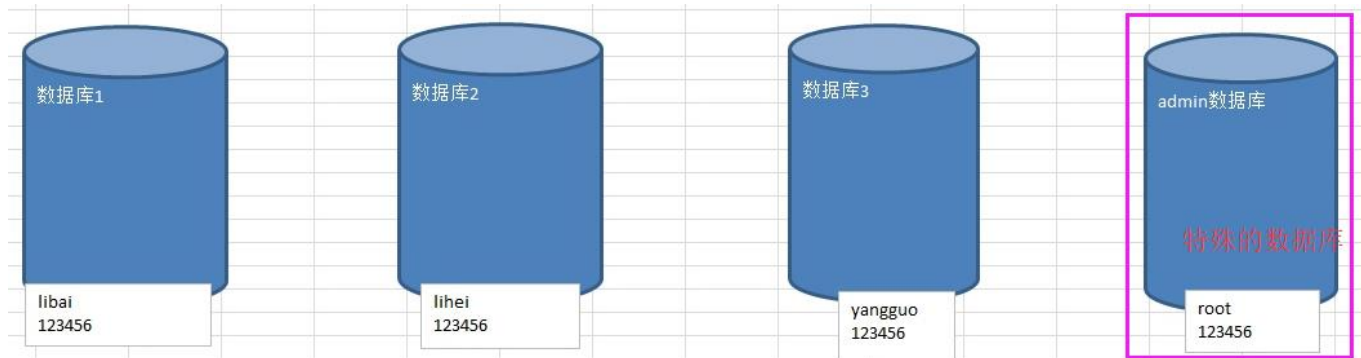
```
C:\wamp\mongodb\bin>mongo localhost:2222
MongoDB shell version: 2.0.3
connecting to: localhost:2222/test
> use stu
switched to db stu
> db.php.find(<)
{ "_id" : ObjectId("58ef37e09d2cac9e181bfc1b"), "name" : "xiaobai", "age" : 12 }
{ "_id" : ObjectId("58ef37f09d2cac9e181bfc1c"), "name" : "xiaohei", "age" : 22 }
>
```

如果有数据，则成功了。

第 11 章 MongoDB 权限、安全

1、权限概述

在 MongoDB 中，用户是属于数据库的，每个数据库都有自己的管理员，管理员登录后，只能操作所属的数据库。但是，在 **admin** 数据库（系统自带的）中创建的用户是超级管理员，登录后可以操作任何的数据库。



熟悉 Oracle 的童鞋们都知道，数据库用户有两种，一种是管理员，用来管理用户，一种是普通用户，用来访问数据。类似的，为 MongoDB 规划用户鉴权时，至少要规划两种角色：用户管理员和数据库用户。如果搭建了分片或主从，可能还会要规划数据库架构管理员的角色，它们专门用来调整数据库的分布式架构。

2、创建用户

2.x 版本的做法（可以不管）：

语法：

1) 选择数据库：use 数据库名称

2) 添加用户：db.addUser(用户名, 密码, 是否只读) // 是否只读，默认是 false，即可读可写，如果是 true，则只读。

注意点：

在创建用户之前，必须先创建一个超级管理员（即在 admin 数据库下创建用户）。

4.x 版本的做法:

mongodb 内置角色（网上复制的）:

(1). 数据库用户角色

针对每一个数据库进行控制。

read: // 提供了读取所有非系统集合，以及系统集合中的system.indexes, system.js, system.namespaces
readWrite: // 包含了所有read权限，以及修改所有非系统集合的和系统集合中的system.js的权限。

(2). 数据库管理角色

每一个数据库包含了下面的数据库管理角色。

dbOwner: // 该数据库的所有者，具有该数据库的全部权限。
dbAdmin: // 一些数据库对象的管理操作，但是没有数据库的读写权限。（参考：<http://docs.mongodb.org/manual/reference/built-in-roles/#dbAdmin>）
userAdmin: // 为当前用户创建、修改用户和角色。拥有userAdmin权限的用户可以将该数据库的任意权限赋予任意的用户，该权限不能操作数据库数据。

(3). 集群管理权限

admin数据库包含了下面的角色，用户管理整个系统，而非单个数据库。这些权限包含了复制集和共享集群的管理函数。

clusterAdmin: // 提供了最大的集群管理功能。相当于clusterManager, clusterMonitor, and hostManager和dropDatabase的权限组合。
clusterManager: // 提供了集群和复制集管理和监控操作。拥有该权限的用户可以操作config和local数据库（即分片和复制功能）
clusterMonitor: // 仅仅监控集群和复制集。
hostManager: // 提供了监控和管理服务器的权限，包括shutdown节点, logrotate, repairDatabase等。
// 备份恢复权限: admin数据库中包含了备份恢复数据的角色。包括backup、restore等。

(4). 所有数据库角色

admin数据库提供了一个mongod实例中所有数据库的权限角色。

readAnyDatabase: // 具有read每一个数据库权限。但是不包括应用到集群中的数据库。
readWriteAnyDatabase: // 具有readWrite每一个数据库权限。但是不包括应用到集群中的数据库。
userAdminAnyDatabase: // 具有userAdmin每一个数据库权限，但是不包括应用到集群中的数据库。
dbAdminAnyDatabase: // 提供了dbAdmin每一个数据库权限，但是不包括应用到集群中的数据库。

(5). 超级管理员权限

root: // dbadmin到admin数据库、useradmin到admin数据库以及UserAdminAnyDatabase。但它不具有备份恢复、直接操作system.*集合的权限，但是拥有root权限的超级用户可以自己给自己赋予这些

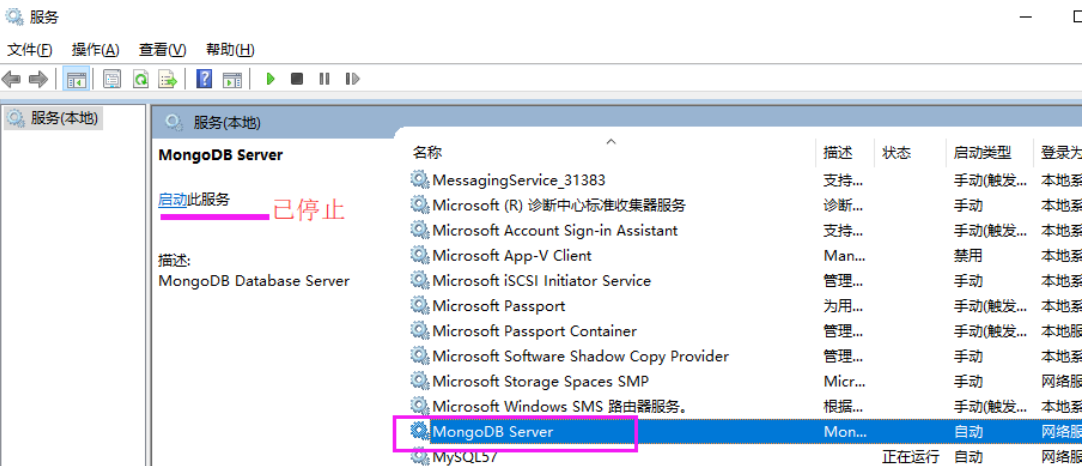
权限。

- (6) . 备份恢复角色: **backup、restore**;
- (7) . 内部角色: **__system** （两个下划线）

创建超级管理员

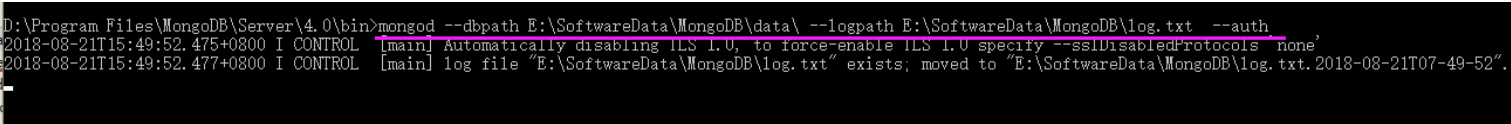
数据库管理员是第一个要创建的用户。在没有创建任何用户之前，你可以随意创建用户；但数据库中一旦有了用户，那么未登录的客户端就没有权限做任何操作了，除非使用 db.auth(username, password) 方法登录。

停止服务



打开认证--auth，重启，这里以开发环境形式，进入 MongoDB 的 bin 目录，执行：。

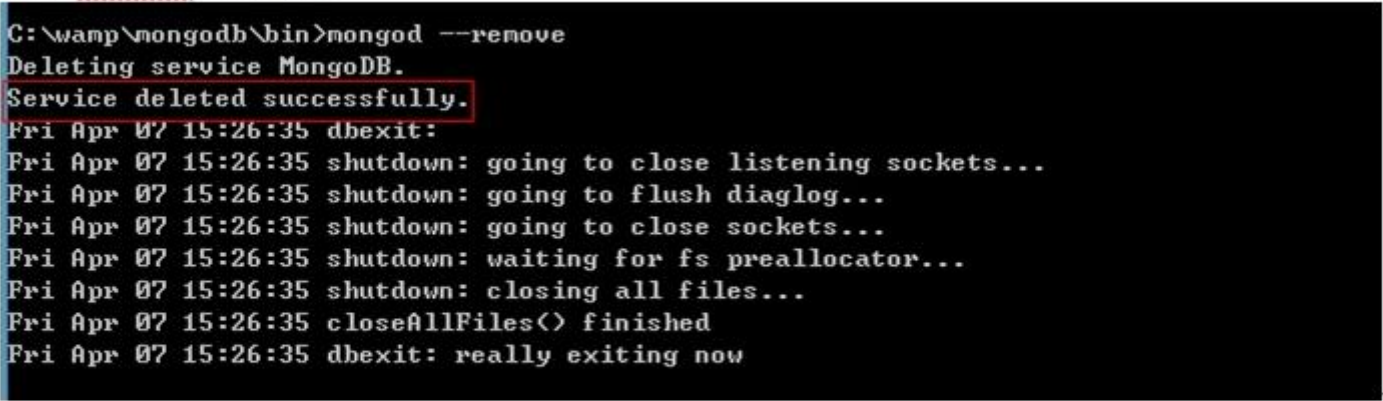
```
mongod --dbpath E:\SoftwareData\MongoDB\data\ --logpath E:\SoftwareData\MongoDB\log.txt --auth
```



该窗口是不能关闭的哈。

如果以前是用生产环境方式启动的，则执行下图两步进行启动。

第二步：以管理员的方式打开 cmd 进入到 mongodb 里面的 bin 目录，运行 `mongod --remove`



第三步：卸载完成服务后，再重新安装，安装添加--auth 选项。

```
mongod --install --logpath d:/wamp/mongodb/log.txt --dbpath d:/wamp/mongodb/data --auth
```

创建超级管理员

```
D:\Program Files\MongoDB\Server\4.0\bin>mongo

MongoDB shell version v4.0.1

connecting to: mongodb://127.0.0.1:27017

MongoDB server version: 4.0.1


// 选择数据库
> use admin

switched to db admin


// 创建用户。role:"userAdminAnyDatabase"等同于超级管理员，但不能操作数据库数据，参考MongoDB内置角色；db:"admin"表示该用户属于admin数据库的
> db.createUser({user:"testadmin", pwd:"123456", roles:[{role:"userAdminAnyDatabase", db:"admin"}]})

Successfully added user: {
  "user" : "testadmin",
  "roles" : [
    {
      "role" : "userAdminAnyDatabase",
      "db" : "admin"
    }
  ]
}


// 报错，提示没有认证
> show dbs

2018-08-21T15:55:36.527+0800 E QUERY [js] Error: listDatabases failed:{
  "ok" : 0,
  "errmsg" : "command listDatabases requires authentication",
  "code" : 13,
  "codeName" : "Unauthorized"
} :
_getErrorWithCode@src/mongo/shell/utils.js:25:13
Mongo.prototype.getDBs@src/mongo/shell/mongo.js:67:1
shellHelper.show@src/mongo/shell/utils.js:876:19
shellHelper@src/mongo/shell/utils.js:766:15
@(shellhelp2):1:1


// 认证成功返回1
> db.auth("testadmin", "123456")

1

> show dbs

admin 0.000GB
config 0.000GB
local 0.000GB

>
```

3、删除用户、修改密码等

获得数据库的所有用户权限信息: db.getUsers()

获得某个用户的权限信息: db.getUser()

创建角色: db.createRole()

更新角色: db.updateRole()

删除角色: db.dropRole()

获得某个角色信息: db.getRole()

删除用户: db.dropUser()

删除所有用户: db.dropAllUsers()

将一个角色赋予给用户: db.grantRolesToUser()

撤销某个用户的某个角色权限: db.revokeRolesFromUser()

更改密码: db.changeUserPassword()

```
> db.getUsers()
[
  {
    "_id": "admin.testadmin",
    "user": "testadmin",
    "db": "admin",
    "roles": [
      {
        "role": "userAdminAnyDatabase",
        "db": "admin"
      }
    ],
    "mechanisms": [
      "SCRAM-SHA-1",
      "SCRAM-SHA-256"
    ]
  }
]
```

结束日期: 2018-08-11

修改开始时间: 2018-08-21

修改结束时间: 2018-08-21