

MONGODB 入门篇

惨绿少年 Linux运维, NoSQL, 数据库

0评论

来源：本站原创

68°C

字体：

小

中

大

1.1 数据

库管理系统



在了解MongoDB之前需要先了解先数据库管理系统

1.1.1 什么是数据？

数据（英语：data），是指未经过处理的原始记录。

一般而言，数据缺乏组织及分类，无法明确的表达事物代表的意义，它可能是一堆的杂志、一大叠的报纸、数种的开会记录或是整本病人的病历纪录。数据描述事物的符号记录，是可定义为意义的实体，涉及事物的存在形式。是关于事件之一组离散且客观的事实描述，是构成讯息和知识的原始材料。

1.1.2 什么是数据库管理系统？

数据库管理系统（英语：database management system，缩写：DBMS）是一种针对对象数据库，为管理数据库而设计的大型电脑软件管理系统。

具有代表性的数据管理系统有：Oracle、Microsoft SQL Server、Access、MySQL及PostgreSQL等。通常数据库管理师会使用数据库管理系统来创建数据库系统。

现代DBMS使用不同的数据库模型追踪实体、属性和关系。在个人电脑、大型计算机和主机上应用最广泛的数据库管理系统是关系型DBMS（relational DBMS）。在关系型数据模型中，用二维表格表示数据库中的数据。这些表格称为关系。

数据库管理系统主要分为两大类：RDBMS、NOSQL

关于RDBMS的更多信息参考：<http://www.cnblogs.com/clsn/category/1131345.html>

1.1.3 常见数据库管理系统？

常见的数据库管理系统，及其排名情况如下：

341 systems in ranking, January 2018								
Rank			DBMS	Database Model	Score			
Jan 2018	Dec 2017	Jan 2017			Jan 2018	Dec 2017	Jan 2017	
1.	1.	1.	Oracle +	Relational DBMS	1341.94	+0.40	-74.78	
2.	2.	2.	MySQL +	Relational DBMS	1299.71	-18.36	-66.58	
3.	3.	3.	Microsoft SQL Server +	Relational DBMS	1148.07	-24.42	-72.89	
4.	4.	↑ 5.	PostgreSQL +	Relational DBMS	386.18	+0.75	+55.81	
5.	5.	↓ 4.	MongoDB +	Document store	330.95	+0.18	-0.96	
6.	6.	6.	DB2 +	Relational DBMS	190.28	+0.70	+7.78	
7.	7.	↑ 8.	Microsoft Access	Relational DBMS	126.70	+0.82	-0.75	
8.	↑ 9.	↓ 7.	Cassandra +	Wide column store	123.88	+0.67	-12.57	
9.	↓ 8.	9.	Redis +	Key-value store	123.14	-0.10	+4.44	
10.	10.	↑ 11.	Elasticsearch +	Search engine	122.55	+2.77	+16.38	
11.	11.	↓ 10.	SQLite +	Relational DBMS	114.25	-0.94	+1.88	
12.	12.	12.	Teradata	Relational DBMS	72.63	-2.11	-1.54	
13.	↑ 14.	13.	SAP Adaptive Server	Relational DBMS	65.46	-0.22	-3.64	
14.	↓ 13.	14.	Solr	Search engine	64.37	-1.93	-3.71	
15.	15.	↑ 16.	Splunk	Search engine	64.00	+0.21	+8.51	
16.	16.	↓ 15.	HBase	Wide column store	61.64	-1.78	+2.50	
17.	17.	↑ 20.	MariaDB +	Relational DBMS	58.30	+1.56	+13.26	
18.	↑ 19.	↑ 19.	Hive +	Relational DBMS	55.49	+0.81	+4.35	
19.	↓ 18.	↓ 17.	FileMaker	Relational DBMS	55.20	+0.00	+1.72	
20.	20.	↓ 18.	SAP HANA +	Relational DBMS	46.16	-0.33	-5.77	

图 – 数据库管理系统使用情况世界排名

数据来源: <https://db-engines.com/en/ranking>

1.2 NoSQL是什么？

1.2.1 NoSQL简介

NoSQL是对不同于传统的关系数据库的数据库管理系统的统称。

两者存在许多显著的不同点，其中最重要的是NoSQL不使用SQL作为查询语言。其数据存储可以不需要固定的表格模式，也经常会避免使用SQL的JOIN操作，一般有水平可扩展性的特征。

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再次提出了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“非关联型的”，强调Key-Value Stores和文档数据库的优点，而不是单纯的反对RDBMS。

基于2014年的收入，NoSQL市场领先企业是MarkLogic，MongoDB和Datastax。基于2015年的人气排名，最受欢迎的NoSQL数据库是MongoDB，Apache Cassandra和Redis。

1.2.2 NoSQL数据库四大家族

NoSQL中的四大家族主要是：列存储、键值、图像存储、文档存储，其类型产品主要有以下这些。

存储类型	NoSQL	
键值存储	最终一致性键值存储	Cassandra、Dynamo、Riak、Hibari、Virtuoso、Voldemort
	内存键值存储	Memcached、Redis、Oracle Coherence、NCache、Hazelcast、Tuple space、Velocity
	持久化键值存储	BigTable、LevelDB、Tokyo Cabinet、Tarantool、TreapDB、Tuple space
文档存储	MongoDB、CouchDB、SimpleDB、Terrastore、BaseX、Clusterpoint、Riak、No2DB	
图存储	FlockDB、DEX、Neo4J、AllegroGraph、InfiniteGraph、OrientDB、Pregel	
列存储	Hbase、Cassandra、Hypertable	

1.2.3 NoSQL的优势

高可扩展性、分布式计算、没有复杂的关系、低成本

架构灵活、半结构化数据

1.2.4 NoSQL与RDBMS对比

NoSQL	RDBMS
代表着不仅仅是SQL	高度组织化结构化数据
没有声明性查询语言	结构化查询语言
没有预定义的模式	(SQL) (SQL)
键-值对存储，列存储，文档存储，图形数据库	数据和关系都存储在单独的表中。
最终一致性，而非ACID属性	

非结构化和不可预知的数据

CAP定理

高性能，高可用性和可伸缩性

数据操纵语言，数据定义语言

严格的一致性

基础事务

1.3 MongoDB简介

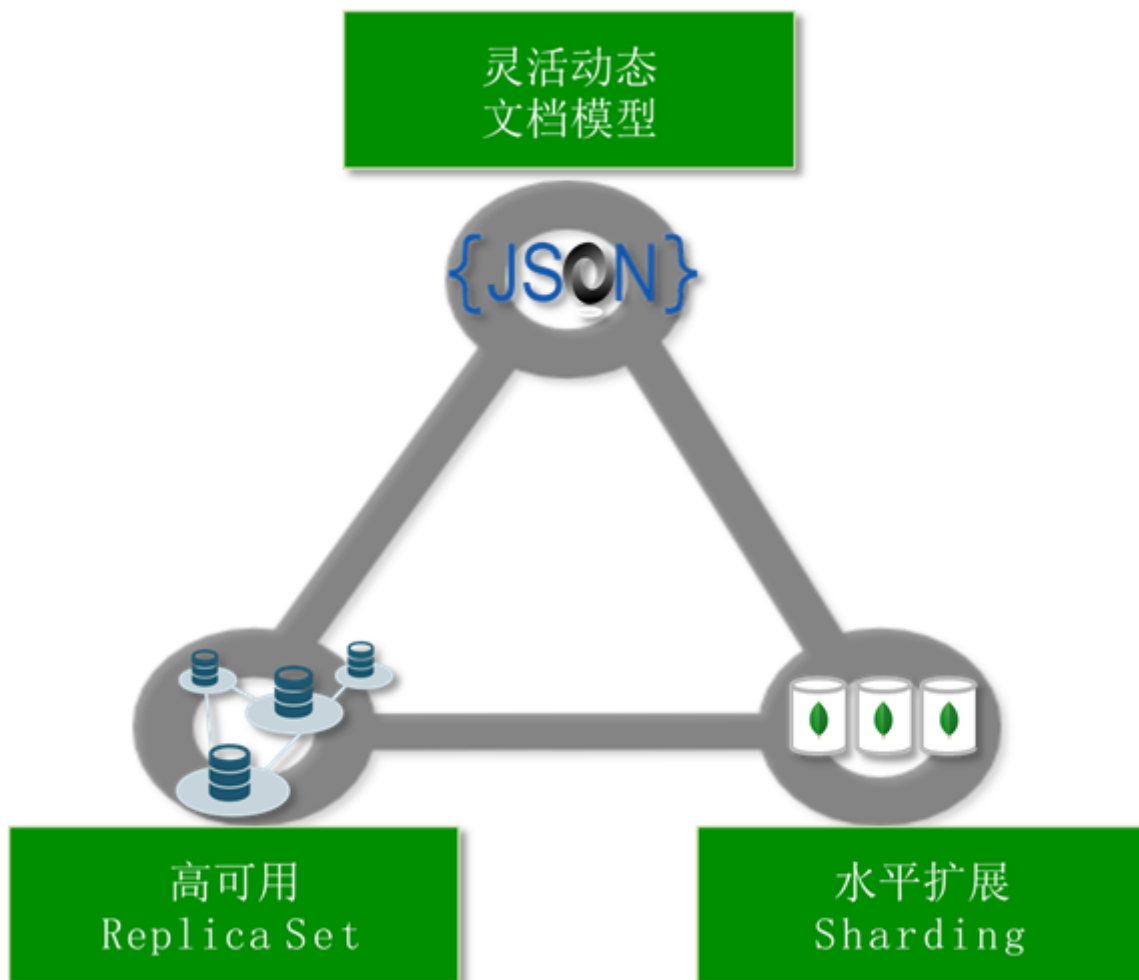
1.3.1 MongoDB是什么



MongoDB并非芒果的意思，而是源于 Humongous（巨大）一词。

1.3.2 MongoDB的特性

MongoDB的3大技术特色如下所示：



除了上图所示的还支持：

二级索引、动态查询、全文搜索、聚合框架、MapReduce、GridFS、地理位置索引、内存引擎、地理分布等一系列的强大功能。

但是其也有些许的**缺点**，例如：

多表关联： 仅仅支持Left Outer Join

SQL 语句支持： 查询为主，部分支持

多表原子事务： 不支持

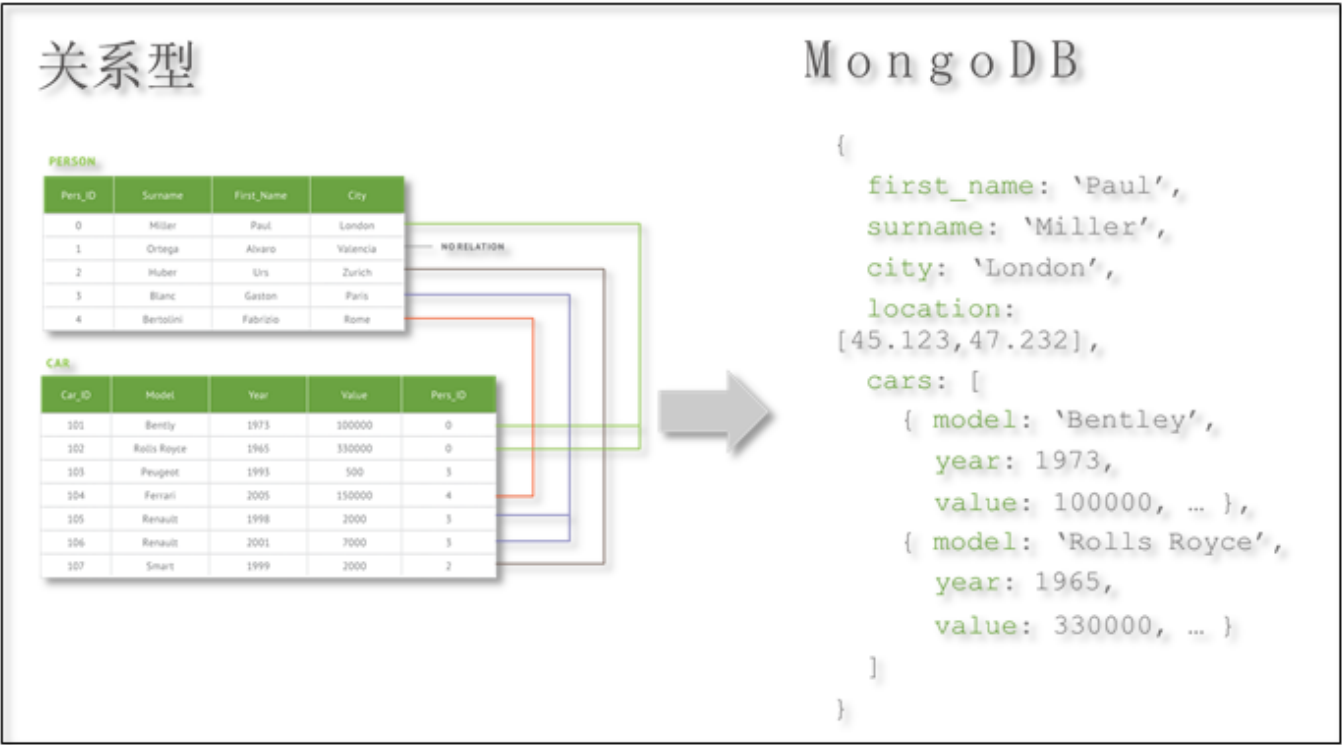
多文档原子事务： 不支持

16MB 文档大小限制，不支持中文排序，服务端 Javascript 性能欠佳

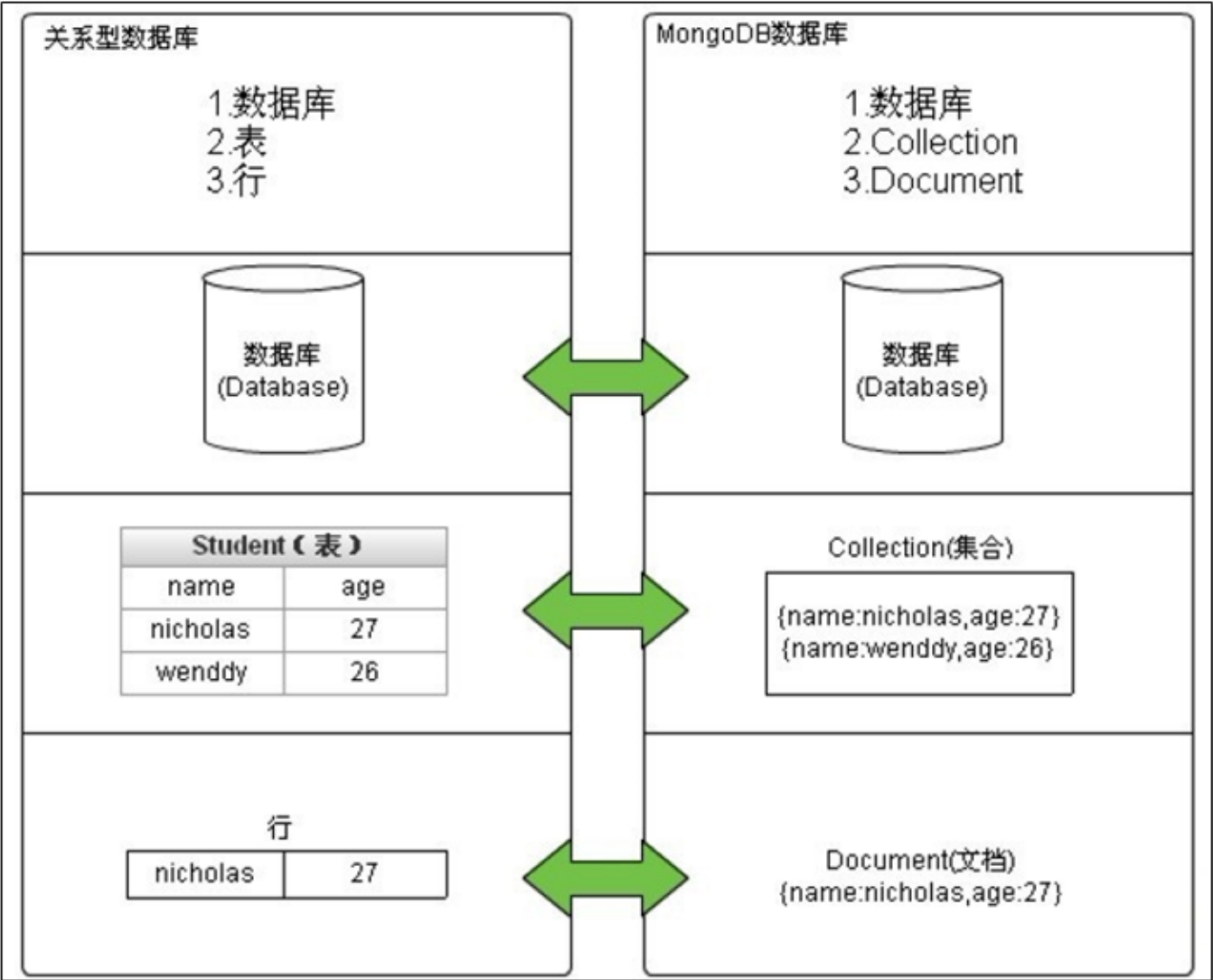
1.3.3 关系型数据库与mongodb对比

存储方式对比

在传统的关系型数据库中，存储方式是以表的形式存放，而在MongoDB中，以文档的形式存在。



数据库中的对应关系，及存储形式的说明



MongoDB与SQL的结构对比详解

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 <code>_id</code> field.
aggregation (e.g. group by)	aggregation pipeline See the SQL to Aggregation Mapping Chart.

1.3.4 MongoDB数据存储格式

JSON格式


JSON 数据格式与语言无关，脱胎于 JavaScript，但目前很多编程语言都支持 JSON 格式数据的生成和解析。JSON 的官方 MIME 类型是 `application/json`，文件扩展名是 `.json`。

MongoDB 使用JSON（JavaScript ObjectNotation）文档存储记录。

JSON数据库语句可以容易被解析。

Web 应用大量使用，NAME-VALUE 配对

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```



BSON格式

BSON是由10gen开发的一个数据格式，目前主要用于MongoDB中，是MongoDB的数据存储格式。BSON基于JSON格式，选择JSON进行改造的原因主要是JSON的通用性及JSON的schemaless的特性。

二进制的JSON，JSON文档的二进制编码存储格式

BSON有JSON没有的Date和BinData

MongoDB中document以BSON形式存放

例如：

```
> db.meeting.insert({meeting:"M1 June",Date:"2018-01-06"});
```

1.3.5 MongoDB的优势

🔗 MongoDB是开源产品

🔗 On GitHub Url: <https://github.com/mongodb>

🔗 Licensed under the AGPL，有开源的社区版本

🔗 起源& 赞助by MongoDB公司，提供商业版licenses 许可

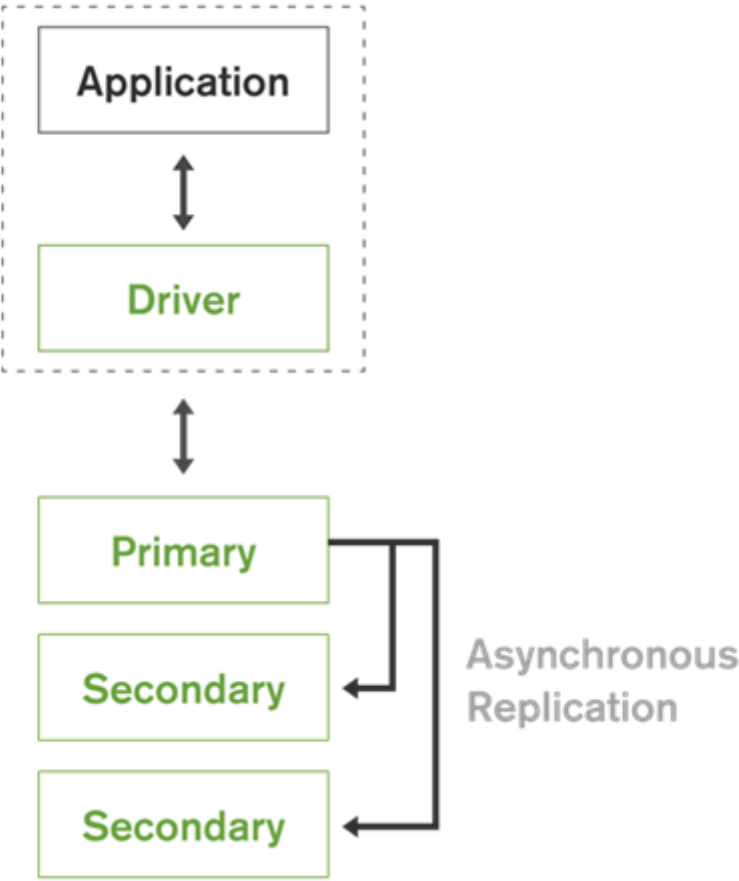
这些优势造就了mongodb的丰富的功能：

JSON 文档模型、动态的数据模式、二级索引强大、查询功能、自动分片、水平扩展、自动复制、高可用、文本搜索、企业级安全、聚合框架MapReduce、大文件存储GridFS

1.3.6 高可用的复制集群

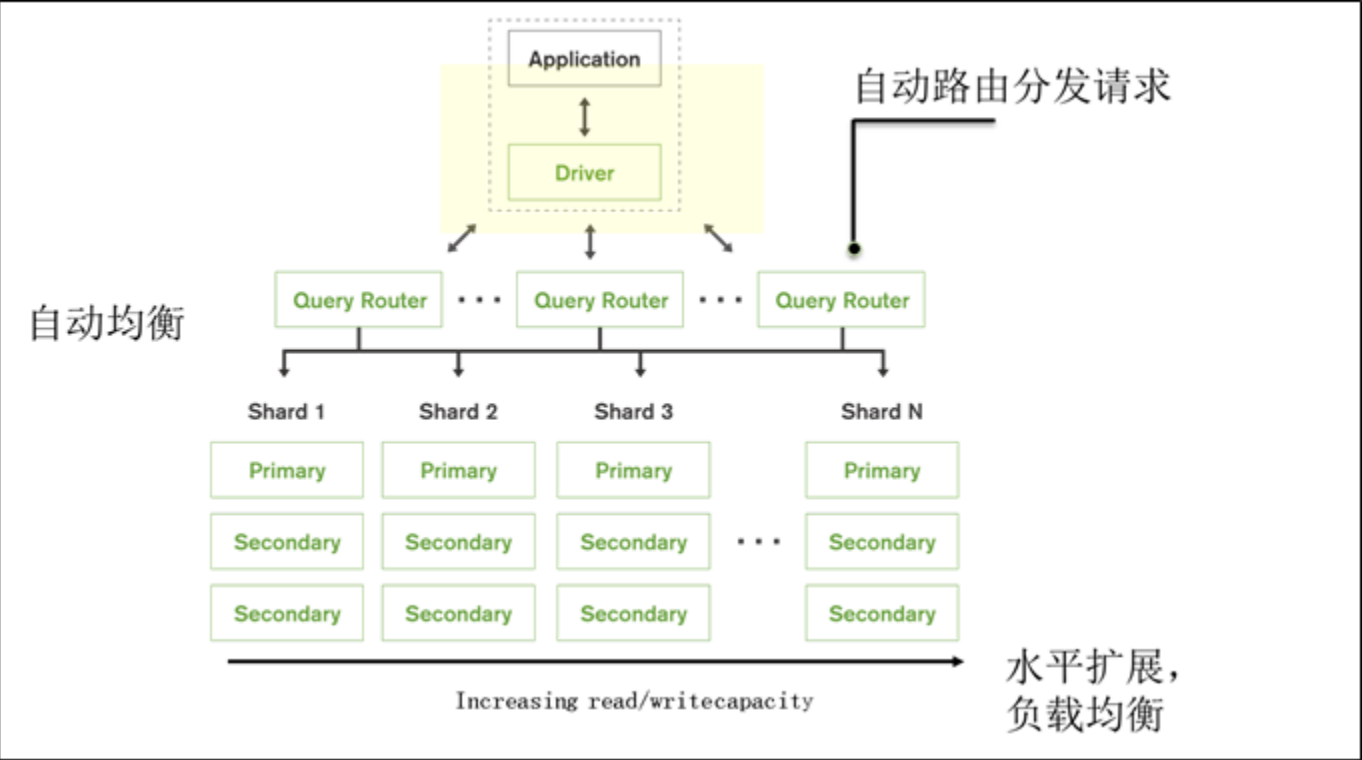
自动复制和故障切换

多数据中心支持滚动维护无需关机支持最多50个成员

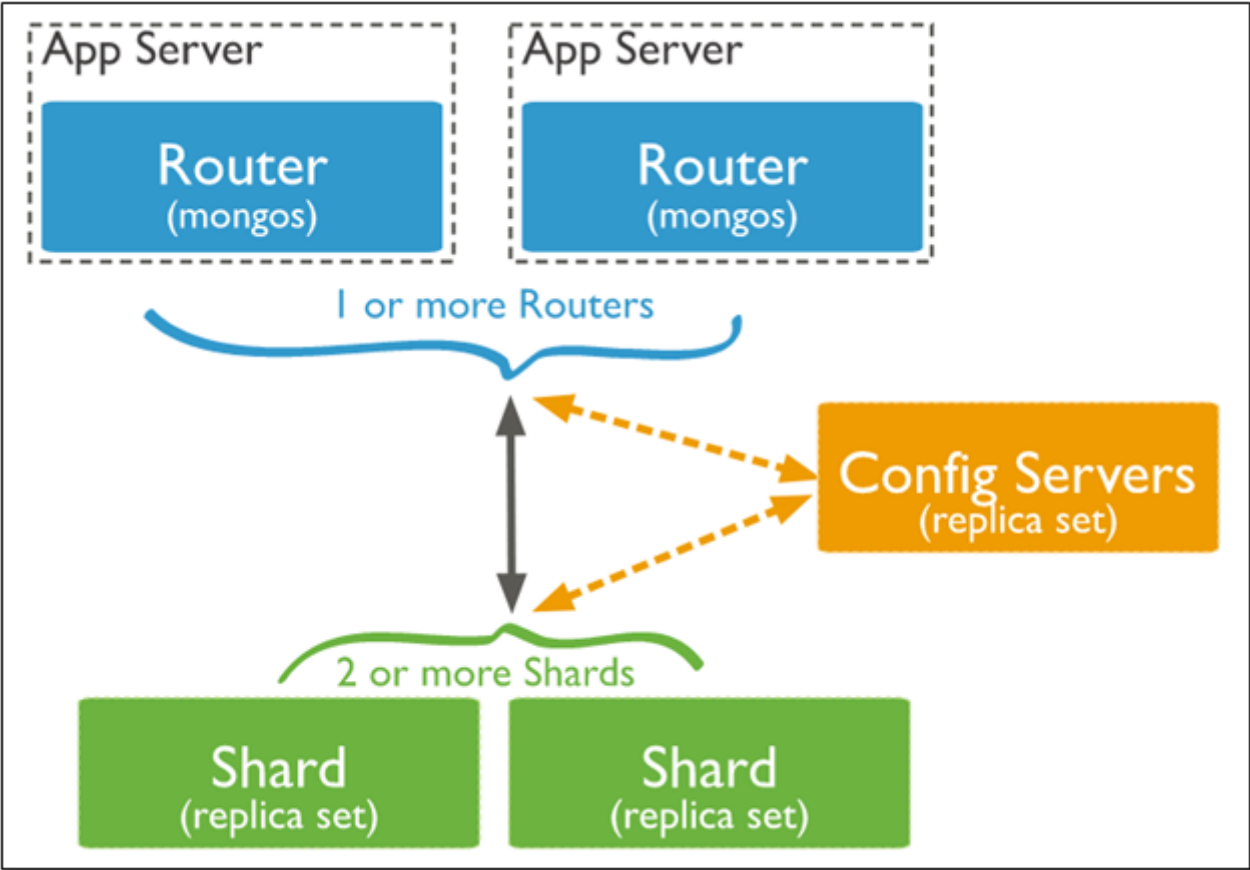


1.3.7 水平扩展

这种方式是目前构架上的主流形式，指的是通过增加服务器数量来对系统扩容。在这样的构架下，单台服务器的配置并不会很高，可能是配置比较低、很廉价的 PC，每台机器承载着系统的一个子集，所有机器服务器组成的集群会比单体服务器提供更强大、高效的系统容量。



这样的问题是系统构架会比单体服务器复杂，搭建、维护都要求更高的技术背景。分片集群架构如下图所示：



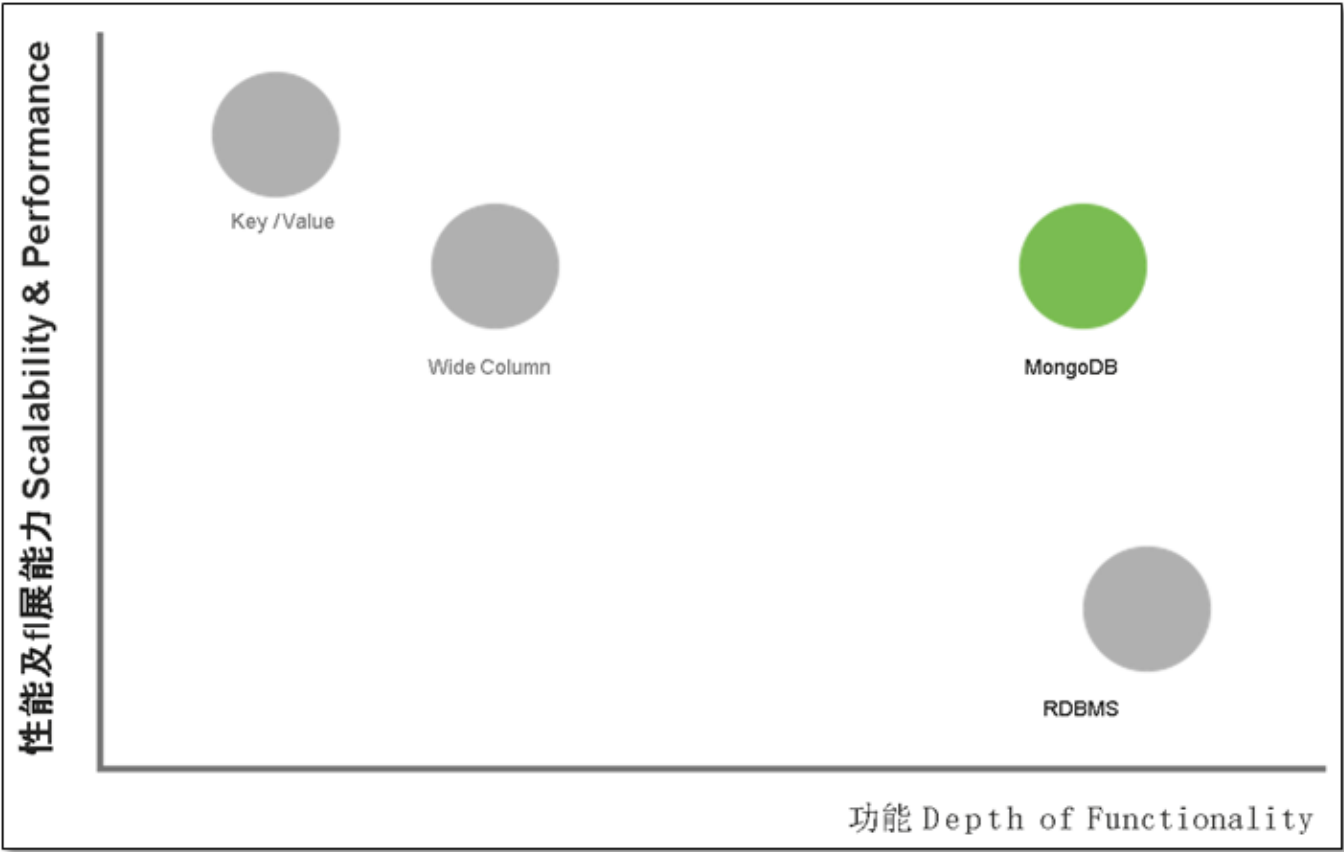
1.3.8 各存储引擎的对比

	MySQL InnoDB	MySQL NDB	Oracle	MongoDB MAPI	MongoDB WiredTiger
事务	YES	YES	ES	NO	NO
锁粒度	ROW-level	ROW-level	ROW-level	Collection-level	Document-level
Geospatial	YES	YES	YES	YES	YES
MVCC	YES	NO	YES	NO	NO
Replication	YES	YES	YES	YES	YES
外键	YES	YES(From 7.3)	YES	NO	NO
数据库集群	NO	YES	YES	YES	YES
B-TREE索引	YES	YES	YES	YES	YES
全文检索	YES	NO	YES	YES	YES

数据压缩	YES	NO	YES	NO	YES
存储限制	64TB	384EB	NO	NO	NO
表分区	YES	YES	YES	YES (分片)	YES (分片)

1.3.9 数据库功能和性能对比

由下图可以看出MongoDB数据库的性能扩展能力及功能都较好，都能够在数据库中，站立一席之地。



1.3.10 MongoDB适用场景

网站数据、缓存等大尺寸、低价值的数据

在高伸缩性的场景，用于对象及JSON数据的存储。



22

1.3.11 MongoDB 慎用场景

慎用场景	原因
PB 数据持久存储大数据分析数据湖	Hadoop、Spark提供更多分析运算功能和工具，并行计算能力更强 MongoDB + Hadoop/Spark
搜索场景：文档有几十个字段，需要按照任意字段搜索并排序及更新操作限制等	不建索引查询太慢，索引太多影响写入及更新操作
ERP、CRM或者类似复杂应用，几十个对象互相关联	关联支持较弱，事务较弱
需要参与远程事务，或者需要跨表、跨文档原子性更新的	MongoDB 事务支持仅限于本机的单文档事务
100% 写可用：任何时间写入不能停	MongoDB换主节点时候会有短暂的时间不可写设计所限

1.3.12 什么时候该MongoDB

应用特征

Yes/No?

我的数据量是有亿万级或者
需要不断扩容

需要2000-3000以上的读写
每秒

新应用，需求会变，数据模
型无法确定

我需要整合多个外部数据源

我的系统需要99.999%高可
用

我的系统需要大量的地理位
置查询

我的系统需要提供最小的
latency

我要管理的主要数据对象<10

在上面的表格中进行选择，但有1个yes的时候：可以考虑MongoDB；当有2个以上yes的时候：不会后悔的选择！

1.4 MongoDB的部署

MongoDB官网：<https://www.mongodb.com/>

CentOS6.X版本软件下载地址：https://www.mongodb.org/dl/linux/x86_64-rhel62

其他版本请到进行官网下载。

1.4.1 安装前准备

在安装之前首先确认该版本软件是否支持你的操作系统。

更多详情查看：<https://docs.mongodb.com/manual/installation/>

Platform	3.6 Community & Enterprise	3.4 Community & Enterprise	3.2 Community & Enterprise	3.0 Community & Enterprise
RHEL/CentOS 6.2 and later	✓	✓	✓	✓
RHEL/CentOS 7.0 and later	✓	✓	✓	✓

1.4.2 环境说明

系统环境说明：

```
[root@MongoDB ~]# cat /etc/redhat-release
CentOS release 6.9 (Final)
[root@MongoDB ~]# uname -r
2.6.32-696.el6.x86_64
[root@MongoDB ~]# /etc/init.d/iptables status
iptables: Firewall is not running.
[root@MongoDB ~]# getenforce
Disabled
[root@MongoDB ~]# hostname -I
10.0.0.152 172.16.1.152
```

软件版本说明

本次使用的mongodb版本为：mongodb-linux-x86_64-3.2.8.tgz

1.4.3 部署MongoDB

在root用户下操作

```
cat >> /etc/rc.local <<'EOF'
if test -f /sys/kernel/mm/transparent_hugepage/enabled; then
    echo never > /sys/kernel/mm/transparent_hugepage/enabled
fi
if test -f /sys/kernel/mm/transparent_hugepage/defrag; then
    echo never > /sys/kernel/mm/transparent_hugepage/defrag
fi
EOF
```

该方法仅限与CentOS系统使用，其他系统关闭参照官方文档：

<https://docs.mongodb.com/manual/tutorial/transparent-huge-pages/>

Transparent Huge Pages (THP)，通过使用更大的内存页面，可以减少具有大量内存的机器上的缓冲区 (TLB) 查找的开销。

但是，数据库工作负载通常对THP表现不佳，因为它们往往具有稀疏而不是连续的内存访问模式。您应该在Linux机器上禁用THP，以确保MongoDB的最佳性能。

创建用户

```
groupadd -g 800 mongod
useradd -u 801 -g mongod mongod
```

修改用户密码

```
echo 123456 |passwd --stdin mongod
```

创建程序目录

```
mkdir -p /application/mongodb/ &&\
cd /application/mongodb/ &&\
mkdir -p bin conf log data
```

下载程序

```
cd /application/mongodb/
wget http://downloads.mongodb.org/linux/mongodb-linux-x86_64-rhel62-3.2.8.tgz
```

解压程序

```
tar xf mongodb-linux-x86_64-3.2.8.tgz
cd mongodb-linux-x86_64-3.2.8/bin/ &&\
cp * /mongodb/bin
```

修改程序属主

```
chown -R mongod:mongod /application/mongodb
```

切换到mongod用户，设置用户环境变量

```
su - mongod
```

```
cat >> .bash_profile <<'EOF'
export PATH=/mongodb/bin:$PATH
EOF
source .bashprofile
```

至此，MongoDB数据库部署完成

1.4.4 管理MongoDB

数据库的启动与关闭

```
启动: mongod --dbpath=/application/mongodb/data --logpath=/application/mongodb/log/mongodb.log -
关闭: mongod --shutdown --dbpath=/application/mongodb/data --logpath=/application/mongodb/log/n
```

参数说明:

参数	参数说明
-dbpath	数据存放路径
-logpath	日志文件路径
-logappend	日志输出方式
-port	启用端口号
-fork	在后台运行
-auth	是否需要验证权限登录(用户名和密码)
-bind_ip	限制访问的ip
-shutdown	关闭数据库

登入数据库

```
[mongod@MongoDB ~]$ mongo
MongoDB shell version: 3.2.8
connecting to: test
>
```

使用配置文件的方式管理数据库:

普通格式配置文件:

```
cd /application/mongodb/conf/
[mongod@MongoDB conf]$ vim mongod1.conf
dbpath=/application/mongodb/data
logpath=/application/mongodb/log/mongodb.log
port=27017
logappend=1
fork=1
```

使用配置文件时的启动与关闭:

```
启动: mongod -f mongod1.conf
关闭: mongod -f mongod1.conf --shutdown
```

YAML格式配置文件 (3.X 版本官方推荐使用)


```
[mongod@MongoDB conf]$ cat mongod.conf
systemLog:
  destination: file
  path: "/application/mongodb/log/mongod.log"
  logAppend: true
storage:
  journal:
    enabled: true
  dbPath: "/application/mongodb/data"
processManagement:
  fork: true
net:
  port: 27017
```

在数据库中关闭数据库的方法

```
shell > mongo
[mongod@MongoDB conf]$ mongo
MongoDB shell version: 3.2.8
connecting to: test
> db.shutdownServer()
shutdown command only works with the admin database; try 'use admin'
> use admin
> db.shutdownServer()
server should be down...
```

注:

mongod进程收到SIGINT信号或者SIGTERM信号，会做一些处理

- ＞ 关闭所有打开的连接
- ＞ 将内存数据强制刷新到磁盘
- ＞ 当前的操作执行完毕
- ＞ 安全停止

切忌kill -9

数据库直接关闭，数据丢失，数据文件损失，修复数据库（成本高，有风险）

使用kill命令关闭进程

```
$ kill -2 PID
原理：-2表示向mongod进程发送SIGINT信号。
或
```

```
$ kill -4 PID
```

原理：-4表示向mognod进程发送SIGTERM信号。

使用脚本管理mongodb服务

注：该脚本可以直接在root用户下运行

田田

```
1 [root@MongoDB ~]# cat /etc/init.d/mongod
2 #!/bin/bash
3 #
4 # chkconfig: 2345 80 90
5 # description: mongodb
6 # by clsn
7 # blog_url http://blog.nmtui.com
8 #####
9
10 MONGODIR=/application/mongodb
11 MONGOD=$MONGODIR/bin/mongod
12 MONGOCONF=$MONGODIR/conf/mongod.conf
13 InfoFile=/tmp/start.mongo
14
15 . /etc/init.d/functions
16
17 status(){
18     PID=`awk 'NR==2{print $NF}' $InfoFile`
19     Run_Num=`ps -p $PID|wc -l`
20     if [ $Run_Num -eq 2 ]; then
21         echo "MongoDB is running"
22     else
23         echo "MongoDB is shutdown"
24         return 3
25     fi
26 }
27
28 start() {
29     status &>/dev/null
30     if [ $? -ne 3 ];then
31         action "启动MongoDB,服务运行中..." /bin/false
32         exit 2
33     fi
34     sudo su - mongod -c "$MONGOD -f $MONGOCONF" >$InfoFile 2>/dev/null
35     if [ $? -eq 0 ];then
36         action "启动MongoDB" /bin/true
37     else
38         action "启动MongoDB" /bin/false
39     fi
40 }
41
42
43 stop() {
44     sudo su - mongod -c "$MONGOD -f $MONGOCONF --shutdown" &>/dev/null
45     if [ $? -eq 0 ];then
46         action "停止MongoDB" /bin/true
47     else
48         action "停止MongoDB" /bin/false
49     fi
```

```
50 }
51
52
53 case "$1" in
54     start)
55         start
56         ;;
57     stop)
58         stop
59         ;;
60     restart)
61         stop
62         sleep 2
63         start
64         ;;
65     status)
66         status
67         ;;
68     *)
69         echo $"Usage: $0 {start|stop|restart|status}"
70         exit 1
71 esac
```

[View Code](#) 脚本管理mongodb服务

1.5 MongoDB的基本操作

Mongodb中关键字种类:

db (数据库实例级别)

db本身

db.connection 数据库下的集合信息

db.collection.xxx(

rs (复制集级别)

sh (分片级别)

1.5.1 查询操作

在客户端指定数据库进行连接: (默认连接本机test数据库)

```
[mongod@MongoDB ~]$ mongo 10.0.0.152/admin
MongoDB shell version: 3.2.8
connecting to: 10.0.0.152/admin
> db
admin
```

查看当前数据库版本

```
> db.version()  
3.2.8
```

切换数据库

```
> use test;  
switched to db test
```

显示当前数据库

```
> db  
test  
> db.getName()  
test
```

查询所有数据库

```
> show dbs;  
clsn  0.000GB  
local 0.000GB  
test  0.000GB  
> show databases;  
clsn  0.000GB  
local 0.000GB  
test  0.000GB
```

查看clsn数据库当前状态

```
> use clsn;  
> db.stats()  
{  
  "db" : "clsn",  
  "collections" : 1,  
  "objects" : 10000,  
  "avgObjSize" : 80,  
  "dataSize" : 800000,  
  "storageSize" : 258048,  
  "numExtents" : 0,  
  "indexes" : 1,  
  "indexSize" : 94208,  
  "ok" : 1  
}
```

查看当前数据库的连接机器地址

```
> db.getMongo()  
connection to 127.0.0.1
```

1.5.2 数据管理

创建数据库

```
> use clsn;
```

说明:

创建数据库:

当use的时候, 系统就会自动创建一个数据库。

如果use之后没有创建任何集合。系统就会删除这个数据库。

删除数据库

```
> show dbs;
clsn   0.000GB
local  0.000GB
test   0.000GB
> use clsn
switched to db clsn
> db.dropDatabase()
{ "dropped" : "clsn", "ok" : 1 }
```

说明:

删除数据库:

如果没有选择任何数据库, 会删除默认的test数据库

创建集合

方法一:

```
> use clsn;
switched to db clsn
> db.createCollection('a')
{ "ok" : 1 }
> db.createCollection('b')
{ "ok" : 1 }
```

查看当前数据下的所有集合

```
> show collections;
a
b
> db.getCollectionNames()
[ "a", "b" ]
```

方法二:

当插入一个文档的时候，一个集合就会自动创建。

```
> use clsn;
switched to db clsn
> db.c.insert({name:'clsn'});
WriteResult({ "nInserted" : 1 })
> db.c.insert({url:'http://blog.nmtui.com'});
WriteResult({ "nInserted" : 1 })
```

查看创建的合集

```
> db.getCollectionNames()
[ "a", "b", "c" ]
```

查看合集里的内容

```
> db.c.find()
{ "_id" : ObjectId("5a4cbcea83ec78b7bea904f8"), "name" : "clsn" }
{ "_id" : ObjectId("5a4cbcfc83ec78b7bea904f9"), "url" : "http://blog.nmtui.com" }
```

重命名集合

```
> db.c.renameCollection("clsn")
{ "ok" : 1 }
> db.getCollectionNames()
[ "a", "b", "clsn" ]
```

删除合集

```
> db.a.drop()
true
> db.getCollectionNames()
[ "b", "clsn" ]
```

插入1w行数据

```
> for(i=0;i<10000;i++){ db.log.insert({"uid":i,"name":"mongodb","age":6,"date":new Date()}); }
WriteResult({ "nInserted" : 1 })
```

查询集合中的查询所有记录

```
> db.log.find()
```

注：默认每页显示20条记录，当显示不下的情况下，可以用it迭代命令查询下一页数据。

```
> DBQuery.shellBatchSize=50;    # 每页显示50条记录
50
app> db.log.findOne()           # 查看第1条记录
app> db.log.count()             # 查询总的记录数
app> db.log.find({uid:1000});   # 查询UUID为1000的数据
```

删除集合中的记录数

```
> db.log.distinct("name")      # 查询去掉当前集合中某列的重复数据
[ "mongodb" ]
> db.log.remove({})            # 删除集合中所有记录
WriteResult({ "nRemoved" : 10000 })
> db.log.distinct("name")
[ ]
```

查看集合存储信息

```
> db.log.stats()               # 查看数据状态
> db.log.dataSize()            # 集合中数据的原始大小
> db.log.totalIndexSize()      # 集合中索引数据的原始大小
> db.log.totalSize()           # 集合中索引+数据压缩存储之后的大小
> db.log.storageSize()         # 集合中数据压缩存储的大小
```

pretty()使用

```
> db.log.find({uid:1000}).pretty()
{
  "_id" : ObjectId("5a4c5c0bdf067ab57602f7c2"),
  "uid" : 1000,
  "name" : "mongodb",
  "age" : 6,
  "date" : ISODate("2018-01-03T04:28:59.343Z")
}
```

1.6 MongoDB中用户管理

MongoDB数据库默认是没有用户名及密码的，即无权限访问限制。为了方便数据库的管理和安全，需创建数据库用户。

1.6.1 用户的权限

用户中权限的说明

权限	说明
Read	允许用户读取指定数据库
readWrite	允许用户读写指定数据库
dbAdmin	允许用户在指定数据库中执行管理函数，如索引创建、删除，查看统计或访问system.profile
userAdmin	允许用户向system.users集合写入，可以找指定数据库里创建、删除和管理用户

clusterAdmin	只在admin数据库中可用，赋予用户所有分片和复制集相关函数的管理权限。
readAnyDatabase	只在admin数据库中可用，赋予用户所有数据库的读权限
readWriteAnyDatabase	只在admin数据库中可用，赋予用户所有数据库的读写权限
userAdminAnyDatabase	只在admin数据库中可用，赋予用户所有数据库的userAdmin权限
dbAdminAnyDatabase	只在admin数据库中可用，赋予用户所有数据库的dbAdmin权限。
root	只在admin数据库中可用。超级账号，超级权限

更多关于用户权限的说明参照：<https://docs.mongodb.com/manual/core/security-built-in-roles/>

用户创建语法

```
{
  user: "",
  pwd: "",
  customData: { },
  roles: [
    { role: "",
      db: "" } | "",
    ...
  ]
}
```

语法说明：

- user字段：用户的名字；
- pwd字段：用户的密码；
- cusomData字段：为任意内容，例如可以为用户全名介绍；
- roles字段：指定用户的角色，可以用一个空数组给新用户设定空角色；

roles 字段：可以指定内置角色和用户定义的角色。

1.6.2 创建管理员用户

进入管理数据库

```
> use admin
```

创建管理用户，root权限

```
db.createUser(  
  {  
    user: "root",  
    pwd: "root",  
    roles: [ { role: "root", db: "admin" } ]  
  }  
)
```

注意：

创建管理员角色用户的时候，必须到admin下创建。

删除的时候也要到相应的库下操作。

查看创建完用户后的collections;

```
> show tables;  
system.users # 用户存放位置  
system.version
```

查看创建的管理员用户

```
> show users  
{  
  "_id" : "admin.root",  
  "user" : "root",  
  "db" : "admin",  
  "roles" : [  
    {  
      "role" : "root",  
      "db" : "admin"  
    }  
  ]  
}
```

验证用户是否能用

```
> db.auth("root","root")
1 # 返回 1 即为成功
```

用户创建完成后在配置文件中开启用户验证

```
cat >>/application/mongodb/conf/mongod.conf<<- 'EOF'
security:
  authorization: enabled
EOF
```

重启服务

```
/etc/init.d/mongod restart
```

登陆测试，注意登陆时选择admin数据库

注意：用户在哪个数据库下创建的，最后加上什么库。

方法一：命令行中进行登陆

```
[mongod@MongoDB ~]$ mongo -uroot -proot admin
MongoDB shell version: 3.2.8
connecting to: admin
>
```

方法二：在数据库中进行登陆验证：

```
[mongod@MongoDB ~]$ mongo
MongoDB shell version: 3.2.8
connecting to: test
> use admin
switched to db admin
> db.auth("root","root")
1
> show tables;
system.users
system.version
```

1.6.3 按生产需求创建应用用户

创建对某库的只读用户

在test库创建只读用户test

```
use test
db.createUser(
{
  user: "test",
  pwd: "test",
  roles: [ { role: "read", db: "test" } ]
}
)
```

测试用户是否创建成功

```
db.auth("test","test")
show users;
```

登录test用户，并测试是否只读

```
show collections;
db.createCollection('b')
```

创建某库的读写用户

创建test1用户，权限为读写

```
db.createUser(
{
  user: "test1",
  pwd: "test1",
  roles: [ { role: "readWrite", db: "test" } ]
}
```

查看并测试用户

```
show users;
db.auth("test1","test1")
```

创建对多库不同权限的用户

创建对app为读写权限，对test库为只读权限的用户

```
use app
db.createUser(
{
  user: "app",
  pwd: "app",
  roles: [ { role: "readWrite", db: "app" },
           { role: "read", db: "test" } ]
}
```

查看并测试用户

```
show users
db.auth("app","app")
```

删除用户

删除app用户：先登录到admin数据库

```
mongo -uroot -proot 127.0.0.1/admin
```

进入app库删除app用户

```
use app
db.dropUser("app")
```

1.6.4 自定义数据库

创建app数据库的管理员：先登录到admin数据库

```
use app
db.createUser(
{
  user: "admin",
  pwd: "admin",
  roles: [ { role: "dbAdmin", db: "app" } ]
}
)
```

创建app数据库读写权限的用户并具有clusterAdmin权限：

```
use app
db.createUser(
{
  user: "app04",
  pwd: "app04",
  roles: [ { role: "readWrite", db: "app" },
{ role: "clusterAdmin", db: "admin" }
]
}
)
```

1.7 SQL与MongoDB语言对比

SQL语言与CRUD语言对照

SQL Schema Statements	MongoDB Schema Statements
CREATE TABLE users (id MEDIUMINT NOT NULL AUTO_INCREMENT, user_id Varchar(30), age Number,	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,

<pre>status char(1), PRIMARY KEY (id))</pre>	<pre>status: "A" })) However, you can also explicitly create a collection: db.createCollection("users")</pre>
<pre>ALTER TABLE users ADD join_date DATETIME</pre>	<p>在Collection 级没有数据结构概念。然而在 document级，可以通过\$set在 update 操作添加列到文档中。</p> <pre>db.users.update({ }, { \$set: { join_date: new Date() } }, { multi: true })</pre>
<pre>ALTER TABLE users DROP COLUMN join_date</pre>	<p>在Collection 级没有数据结构概念。然而在 document级，可以通过\$unset 在update操作从文档中删除列。</p> <pre>db.users.update({ }, { \$unset: { join_date: "" } }, { multi: true })</pre>
<pre>CREATE INDEX idx_user_id_asc ON users(user_id)</pre>	<pre>db.users.createIndex({ user_id: 1 })</pre>

CREATE INDEX idx_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()

插入/删除/更新 语句对比

SQL Statements	MongoDB Statements
INSERT INTO users(user_id, age status) VALUES ("bcd001", 45, "A")	db.users.insert({ user_id: "bcd001", age: 45, status: "A" })
DELETE FROM users WHERE status = "D"	db.users.remove({ status: "D" })
DELETE FROM users	db.users.remove({})
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 })

查询类操作对比

SQL Statements	SELECT MongoDB find() Statements
SELECT * FROM users	db.users.find()
SELECT id, user_id, status FROM users	db.users.find({ }, { user_id: 1, status: 1, _id: 0 })
SELECT user_id, status FROM users	db.users.find({ }, { user_id: 1, status: 1 })
SELECT * FROM users WHERE status = "A"	db.users.find({ status: "A" })
SELECT user_id, status FROM users WHERE status = "A"	db.users.find({ status: "A" }, { user_id: 1, status: 1, _id: 0 })

1.8 错误解决

在登陆数据库的时候，发现会由描述文件相关的报错。

```
[mongod@MongoDB mongod]$ mongo
MongoDB shell version: 3.2.8
connecting to: test
Server has startup warnings:
2018-01-03T11:08:55.526+0800 I CONTROL [initandlisten]
2018-01-03T11:08:55.526+0800 I CONTROL [initandlisten] ** WARNING: soft rlimits too low. rlimit
```

解决办法：

```
cat >> /etc/security/limits.conf <<EOF
mongod    soft    nofile    32767.5
mongod    soft    nproc     32767.5
EOF
```

修改后，重启服务器，即可解决该问题。

1.9 参考文献

- [1] <https://docs.mongodb.com/manual/introduction/>
- [2] <http://www.mongoing.com/docs/introduction.html>
- [3] <https://zh.wikipedia.org/>
- [4] <https://docs.mongodb.com/manual/core/security-built-in-roles/>

赞0

如无特殊说明，文章均为本站原创，转载请注明出处

- 转载请注明来源：MongoDB 入门篇
- 本文永久链接地址：<https://www.nmtui.com/clsn/lx268.html>

该文章由 惨绿少年 发布



惨绿少年Linux www.nmtui.com