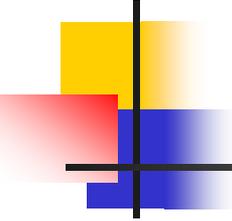
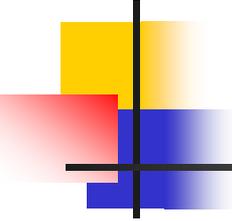


MongoDB高级培训



课程大纲

- 1.MongoDB概述
- 2.MongoDB逻辑视图
- 3.MongoDB物理存储
- 4.MongoDB系统架构
- 5.MongoDB shell
- 6.MongoDB管理和维护



第1章 MongoDB概述

本章导读

- 1.概念
- 2.技术特点
- 3.应用场景

1.1 MongoDB概念

■ 概述

MongoDB是一个强大、灵活、可扩展的数据存储方式，它扩展了关系型数据的众多有用功能，比如辅助索引、范围查询和排序，提供了丰富的功能，内置了对MapReduce式聚合的支持，以及对地理空间索引的支持。

MongoDB是面向文档的数据库，而非传统的关系型数据库。关于它的定位，可以描述为：

最像关系型数据库的非关系型数据库

1.2 MongoDB技术特点

■ 无模式：文档的键不会事先定义也不会固定不变。

因为没有模式需要更改，通常不需要迁移大量数据。不必将所有数据都放到一个模式里，应用层可以处理新增或者丢失的键。这样开发者就可以非常容易的变更数据模型。

(1) 无值得情况的处理。

(2) 新增加域的处理：多个网卡，多硬盘，多内存。-采用传统的数据库表冗余接设计无法应对变化。

1.2 MongoDB技术特点

■ 容易扩展

服务器单点部署的瓶颈分析：

→ 内存不够

→ CPU处理能力不足

→ 磁盘IO读写速度极限

→ 网络IO瓶颈

硬件升级VS集群部署

1.2 MongoDB技术特点

■ 功能丰富

- 索引：支持通用辅助索引，能进行快速查询
- 存储Java Script
- 聚合-支持MapReduce和其他聚合工具
- 固定集合
- GridFile：支持大文件存储

不支持：连接（Join）和复杂的多行事务

1.2 MongoDB技术特点

■ 性能可靠

- 基于C++实现，直接操纵内存和IO区别于JVM虚拟机模式
- 采用MongoDB协议Bson作为与服务器交互的主要方式。
- 文档动态填充。
- 预分配数据文件，用空间换性能。

1.2 MongoDB技术特点

■ 管理简便

→ Web UI <http://localhost:18027>(服务端口增加1000)

→ MongoDB shell

MongoDB通过服务器自治来简化数据库的管理，支持主从双机热备。当主服务器宕机后，MongoDB会自动切换到备份服务器上，并且将备份服务器作为活跃服务器。在分布式环境下，集群只需要知道有新增节点增加，就会自动集成和配置新节点。

1.3 MongoDB应用场景

■ 应用场景

→ 动态模式：无固定表结构的数据存储

→ 大数据：**TB**级数据存储，已突破传统关系型数据库存储极限的应用。设备配置再高，终有其性能瓶颈。

第2章 MongoDB逻辑视图

本章导读

- 1.逻辑架构
- 2.服务组件
- 3.驱动程序

Application

MMP



火龙果 · 整理
uml.org.cn

Mongo service

Mongo dump

Mongo restore

Mongo import

Mongo export

Mongo stat

服务与支撑组件

C++ api

Java api

Web ui

Mongo shell

mongos

mongos

mongos

TCP/IP

mongod

mongod

configdb

数据资源

Local

NFS

DFS扩展中

IT基础设施

质量

安全

治理

2.2 服务组件

■ 服务组件

→ mongod

→ mongos

→ mongo shell

→ mongodump

→ mongorestore

→ mongoimport

→ mongoexport

→ mongostat

→ web server

2.3 驱动程序

- 驱动程序

 - C/C++ api

 - Java driver

第3章 MongoDB物理存储

本章导读

- [1.Document](#)
- [2.Collection](#)
- [3.DataBase](#)
- [4.Shard](#)
- [5.Config](#)

3.1 Document

■ JSON数据类型6种

→ null

→ 布尔

→ 数字

→ 字符串

→ 数组

→ 对象

区别于Hbase的纯Bytes类型

3.1 Document

■ BSON数据类型17种

→ null: {"x":null}

→ 布尔: {"x":true}

→ 数字:32、64位整数; 64位浮点数

→ 字符串: {"x":"foobar"}

→ 符号: ??

→ 对象id:12字节, 文档的唯一id, 支持自动生成

→ 日期: {"x":new date()}

→ 正则表达式: {"x":/foobar/i}

→ 代码: {"x":function(){...}} Javascript代码

3.1 Document

■ BSON数据类型17种

- 二进制数据：图像文件二进制值，**shell**无法使用
- 最大值：**shell**不支持
- 最小值：**shell**不支持
- 未定义：{"x":undefine}区别于null
- 数组：{"x":["a","b","c"]}
- 内嵌文档：{"x":{"foo":"bar"}}

3.1 Document

Document

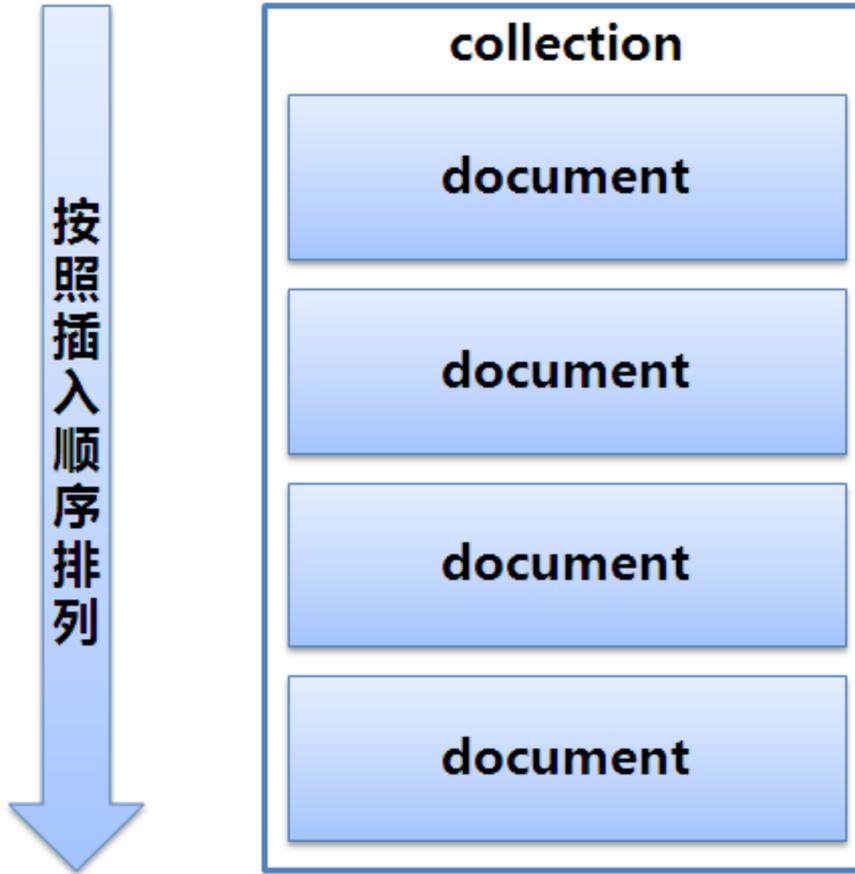
→ `_id`

- (1) 任意类型;
- (2) 集合内唯一;
- (3) 集合间可重复;
- (4) `objectid`: `_id`默认类型
- (5) `_id`支持自动填充



3.2 Collections

collections



如果没有进行分片，按照数据插入顺序排序

3.2 Collections

示例

```
> db.foo.find()
{ "_id" : ObjectId("5249198fe5c43f2ce2ac8afe"), "bar" : "baz" }
{ "_id" : ObjectId("52491a91e5c43f2ce2ac8aff"), "ar" : "az" }
> db.foo.insert(<"cr":"cz">)
> db.foo.insert(<"dr":"dz">)
> db.foo.find()
{ "_id" : ObjectId("5249198fe5c43f2ce2ac8afe"), "bar" : "baz" }
{ "_id" : ObjectId("52491a91e5c43f2ce2ac8aff"), "ar" : "az" }
{ "_id" : ObjectId("52491afee5c43f2ce2ac8b00"), "cr" : "cz" }
{ "_id" : ObjectId("52491b12e5c43f2ce2ac8b01"), "dr" : "dz" }
```

3.3 DataBase

DataBase



3.3 DataBase

■ DataBase

→.ns

→foo.0

→foo.1

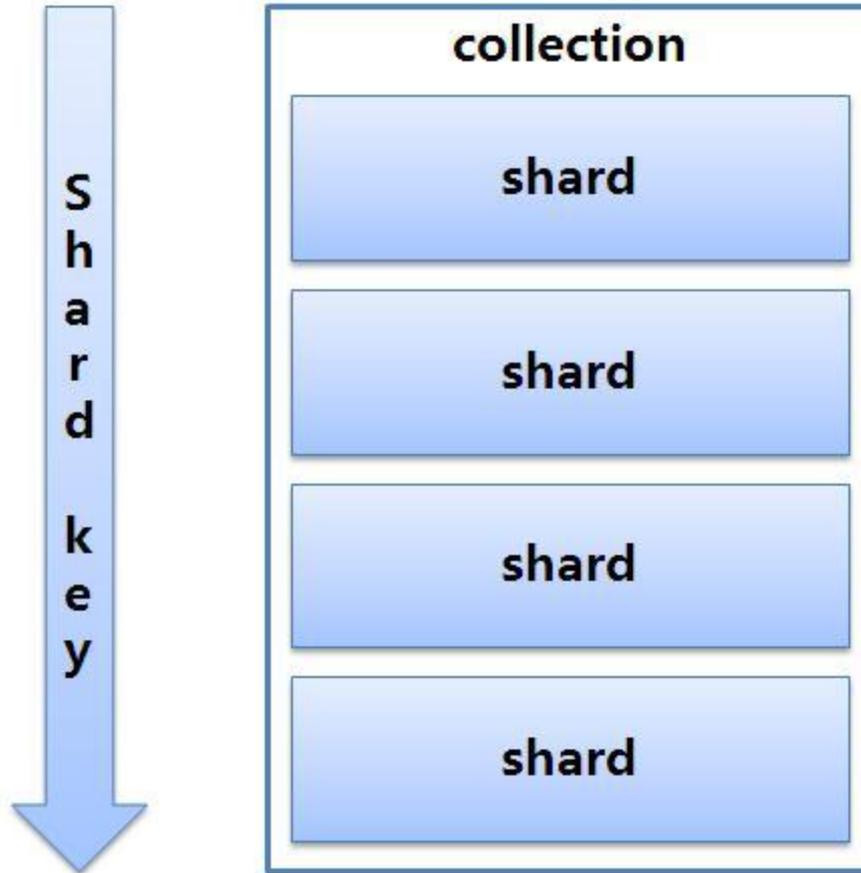
→foo.2

数据库foo会被放在以上的文件中，每个新的数字结尾的数据文件大小会加倍，**直达到达到最大值2G**，这是为了让小数据库不浪费太多的磁盘空间，同时让大数据使用磁盘上连续地空间。

MongoDB为了保证性能还会**预分配数据文件**，可以用—`noprealloc`关闭这一功能。

3.4 Shard

shard



片键作为数据拆分的依据，分片内文档按照片键排序

3.4 Shard

■ 示例

假设文档集合表示人员。如果选择名字（“name”）作为片键，第一片可能会存放名字以**A-F**开头的文档，第二片存的**G-P**的名字，第三片存**Q-Z**的名字。

■ 说明

1. 一个shard上可以存储多个数据库的不同集合
2. 一个shard = mongod服务器

3.5 Config

config

```
> use config
switched to db config
> show dbs
admin    <empty>
book     0.03125GB
config   0.03125GB
login    0.03125GB
user     0.03125GB
> show collections
chunks
collections
databases
lockpings
locks
mongos
settings
shards
system.indexes
version
> db.shards.find(<
< "_id" : "shard0000", "host" : "127.0.0.1:10000" }
< "_id" : "shard0001", "host" : "127.0.0.1:20000" }
< "_id" : "shard0002", "draining" : true, "host" : "127.0.0.1:27017" }
```

3.5 Config

■ Config

- admin用户
- chunks
- shards
- collections
- Databases
- settings
- system.index
- locks
- mongos
- version

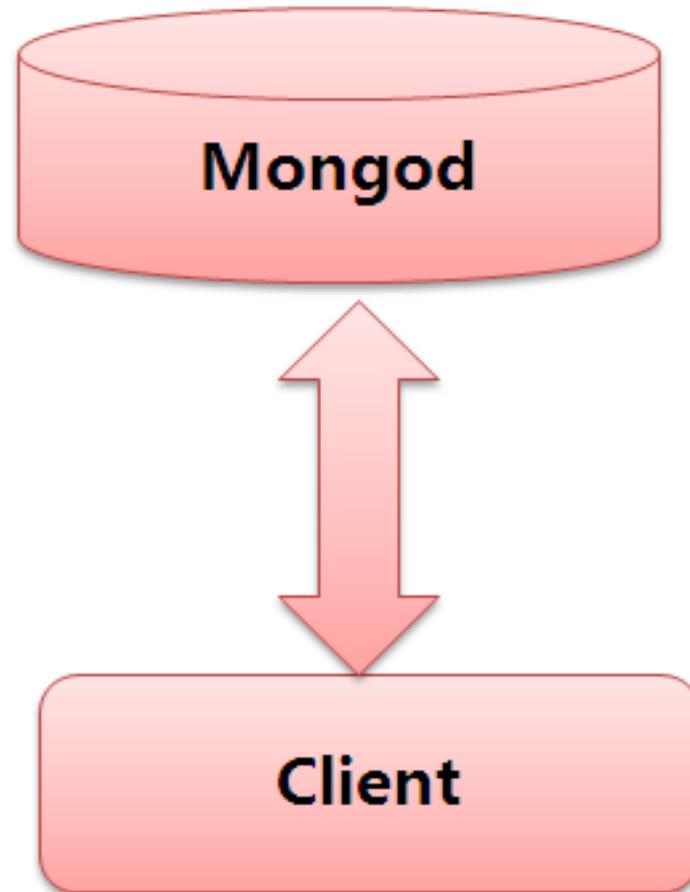
第3章 MongoDB系统架构

本章导读

- 1.物理部署
- 2.逻辑架构
- 3.Client
- 4.mongos
- 5.mongod
- 6.config

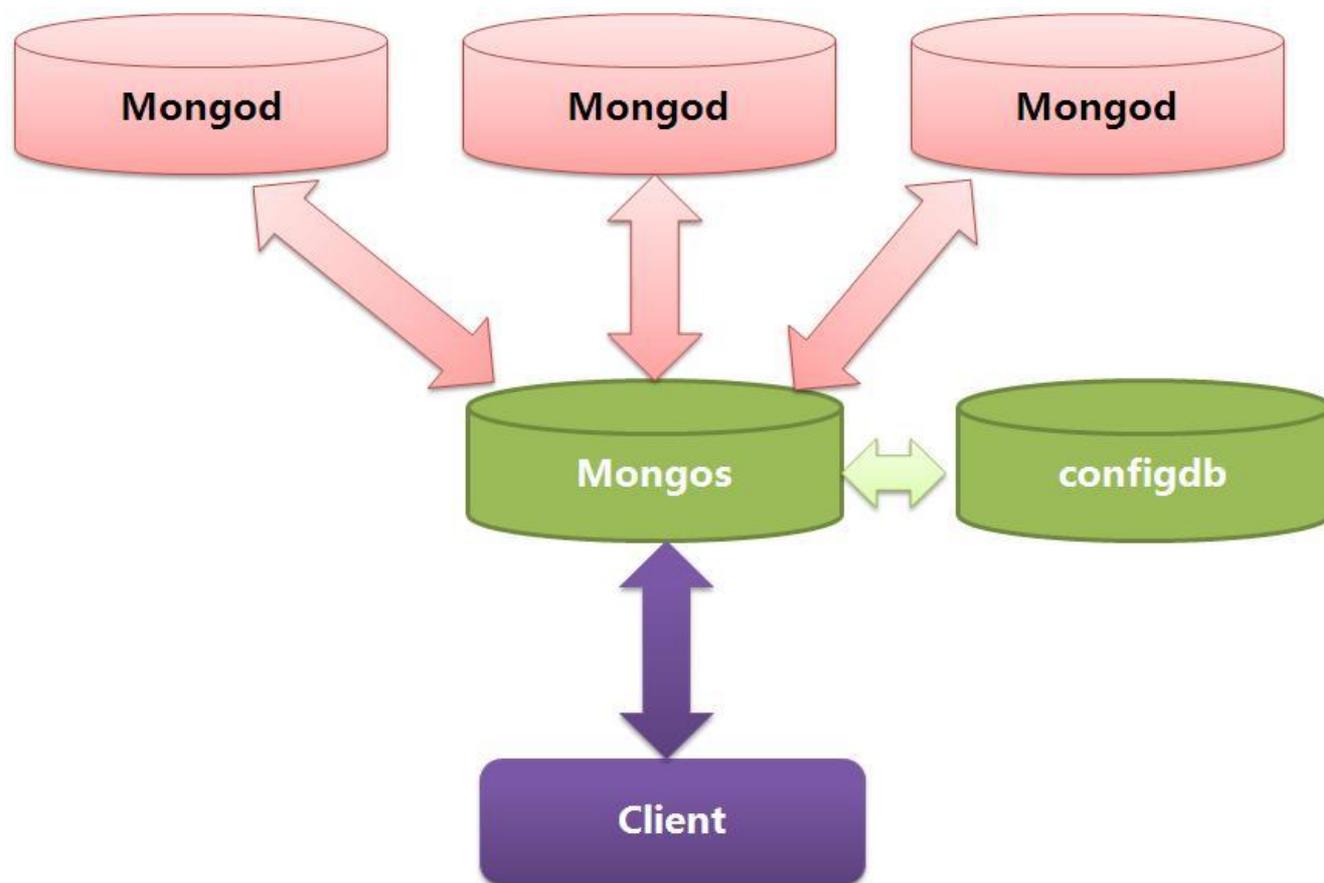
4.1 物理部署

■ 单机模式



4.1 物理部署

■ 分片模式



Application

MMP



火龙果 · 整理
uml.org.cn

Mongo service

Mongo dump

Mongo restore

Mongo import

Mongo export

Mongo stat

服务与支撑组件

C++ api

Java api

Web ui

Mongo shell

mongos

mongos

mongos

TCP/IP

mongod

mongod

configdb

数据资源

Local

NFS

DFS扩展中

IT基础设施

质量

安全

治理

4.3 Client



Client

包括mongodb自己提供的客户端mongo shell和基于mongodb api接口实现的用户自己的应用程序。

4.4 Mongos

■ mongos

1. 单点部署时，用不到；
2. 集群部署时，作为路由服务器存在；
3. 支持多mongos并存；
4. 建议配置一个应用对应一个mongos；
5. 集群部署时，mongos直连client；

4.5 Mongod

■ mongod

- 1.单点部署时，直接作为数据库存储服务器；
- 2.集群部署时，作为切片服务器存在；
- 3.一个mongod就是一个切片；
- 4.每个切片可以存储多个集合；

第5章 MongoDB Shell

本章导读

- 1. 文档操作
- 2. 集合操作
- 3. 数据库操作
- 4. 分片

5.1 文档操作

■ 插入

→ `db.foo.insert({"bar": "baz"})`

→ `_id`键会自动增加;

→ 单个文档最大4M

5.1 文档操作

■ 批量插入

- 传递一个由文档构成的数组给数据库;
- 一次批量插入对应一个**TCP**请求，可以避免多次零碎请求所带来的开销;
- 批量插入当前版本单次最多支持**16M**（**1.8.x**版本）

5.1 文档操作

■ 删除

→ `db.foo.remove()` : 删除全部

→ `db.foo.remove({"opt-out":true})`: 删除所有opt-out为true的文档。

→ 永久性；不可撤销。

→ 删除集合比所有文档性能高很多；

5.1 文档操作

■ 更新

→ `db.foo.update({"name": "joe"}, joe)`

两个参数：

`{"name": "joe"}`-查询条件

Joe: 修改器（目标值）

→ 如果对应该条件有多条记录，将会报错；建议更新时确保更新总是指向唯一文档，可以通过 `_id` 键进行匹配

5.1 文档操作

■ 查询

→ `db.foo.find({"name":"joe"})` : 指定条件查询

→ `db.foo.find()`: 查询全部

→ 查询操作只能针对常量进行，不能够在查询条件中使用其他的键，如下就是错误的：

```
db.stock.find({"in_stock":"this.num_sold"})
```

→ 条件查询：`$lt` (小于)、`$lte`(小于等于)、`$gt`、`$gte`、`$ne` (不等于)

5.1 文档操作

■ 查询

→ `db.foo.find({"name":"joe"})` : 指定条件查询

→ `db.foo.find()`: 查询全部

→ 查询操作只能针对常量进行，不能够在查询条件中使用其他的键，如下就是错误的：

```
db.stock.find({"in_stock":"this.num_sold"})
```

→ 比较运算：`$lt` (小于)、`$lte`(小于等于)、`$gt`、`$gte`、`$ne` (不等于)

5.1 文档操作

■ 查询

→ 复合条件

\$in、\$or、\$not、\$mod

→ 按类型查询

null、正则表达式、数组（\$all、\$size、\$slice）

→ \$where

→ 游标

5.2 集合操作

■ 文档操作

→ show collections

显示数据库中所有集合清单

→ db.collection.help()

```
> db.collection.help()
```

```
DBCollection help
```

```
db.collection.find().help() - show DBCursor help
```

```
db.collection.count()
```

```
db.collection.dataSize()
```

```
db.collection.distinct< key > - eg. db.collection.distinct< 'x' >
```

```
db.collection.drop() drop the collection
```

```
db.collection.dropIndex<name>
```

```
db.collection.dropIndexes()
```

```
db.collection.ensureIndex<keypattern[,options]> - options is an object w
```

```
ith these possible fields: name, unique, dropDups
```

```
db.collection.reIndex()
```

```
db.collection.find<[query],[fields]> - query is an optional query filter
```

```
. fields is optional set of fields to return.
```

```
e.g. db.collection.find<
```

```
x:??> , {name:1, x:1} >
```

```
db.collection.find(...).count()
```

```
db.collection.find(...).limit(n)
```

```
db.collection.find(...).skip(n)
```

```
db.collection.find(...).sort(...)
```

```
db.collection.findOne<[query]>
```

```
db.collection.findAndModify< { update : ... , remove : bool [, query: <
```

```
, sort: <>, 'new': false] >
```

```
db.collection.getDB() get DB object associated with collection
```

```
db.collection.getIndexes()
```

```
db.collection.group< { key : ... , initial: ... , reduce : ...[, cond: ...
```

```
] > >
```

```
db.collection.mapReduce< mapFunction , reduceFunction , <optional params
```

```
> >
```

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

```
db.collection.renameCollection< newName , <dropTarget> > renames the col
```

```
lection.
```

```
db.collection.runCommand< name , <options> > runs a db command with the
```

```
given name where the first param is the collection name
```

```
db.collection.save<obj>
```

```
db.collection.stats()
```

```
db.collection.storageSize() - includes free space allocated to this coll
```

```
ection
```

```
db.collection.totalIndexSize() - size in bytes of all the indexes
```

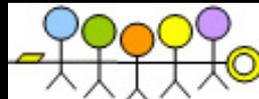
```
db.collection.totalSize() - storage allocated for all data and indexes
```

```
db.collection.update<query, object[, upsert_bool, multi_bool]
```

```
db.collection.validate() - SLOW
```

```
db.collection.getShardVersion() - only for use with sharding
```

```
>
```



火龙果·整理

uml.org.cn

5.3 数据库操作

■ 数据库操作

→ use databasename

→ db

→ shows db

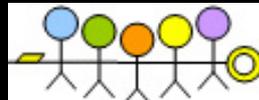
→ db.help()

> db.help()

DB methods:

```
db.addUser(username, password[, readOnly=false])
db.auth(username, password)
db.cloneDatabase(fromhost)
db.commandHelp(name) returns the help for the command
db.copyDatabase(fromdb, todb, fromhost)
db.createCollection(name, { size : ..., capped : ..., max : ... } )
db.currentOp() displays the current operation in the db
db.dropDatabase()
db.eval(func, args) run code server-side
db.getCollection(cname) same as db['cname'] or db.cname
db.getCollectionNames()
db.getLastErrorMessage() - just returns the err msg string
db.getLastErrorMessageObj() - return full status object
db.getMongo() get the server connection object
db.getMongo().setSlaveOk() allow this connection to read from the nonmaster member of a replica pair
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.isMaster() check replica primary status
db.killOp(opid) kills the current operation in the db
db.listCommands() lists all the db commands
db.printCollectionStats()
db.printReplicationInfo()
db.printSlaveReplicationInfo()
db.printShardingStatus()
db.removeUser(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.setProfilingLevel(level, <slowms>) 0=off 1=slow 2=all
db.shutdownServer()
db.stats()
db.version() current version of the server
db.getMongo().setSlaveOk() allow queries on a replication slave server
```



火龙果·整理
uml.org.cn

5.4 分片操作

■ 分片操作

```
>db.runCommand({addshard:"IP:PORT",allowLocal:true})
```

```
>db.runCommand({removeshard:"IP:PORT"})
```

```
>db.shards.find():use config
```

```
>db.databases.find():use config
```

```
> db.shards.help()
```

```
DBCollection help
```

```
db.shards.find().help() - show DBCursor help
```

```
db.shards.count()
```

```
db.shards.dataSize()
```

```
db.shards.distinct< key > - eg. db.shards.distinct< 'x' >
```

```
db.shards.drop() drop the collection
```

```
db.shards.dropIndex<name>
```

```
db.shards.dropIndexes()
```

```
db.shards.ensureIndex<keypattern[,options]> - options is an object with  
these possible fields: name, unique, dropDups
```

```
db.shards.reIndex()
```

```
db.shards.find<[query],[fields]> - query is an optional query filter. fi  
elds is optional set of fields to return.
```

```
e.g. db.shards.find< {x:??
```

```
> , {name:1, x:1} >
```

```
db.shards.find(...).count()
```

```
db.shards.find(...).limit(n)
```

```
db.shards.find(...).skip(n)
```

```
db.shards.find(...).sort(...)
```

```
db.shards.findOne<[query]>
```

```
db.shards.findAndModify< { update : ... , remove : bool [, query: {}], so  
rt: {} , 'new': false } >
```

```
db.shards.getDB() get DB object associated with collection
```

```
db.shards.getIndexes()
```

```
db.shards.group< { key : ... , initial: ... , reduce : ...[, cond: ...] >
```

```
>
```

```
db.shards.mapReduce< mapFunction , reduceFunction , <optional params> >
```

```
db.shards.remove<query>
```

```
db.shards.renameCollection< newName , <dropTarget> > renames the collect  
ion.
```

```
db.shards.runCommand< name , <options> > runs a db command with the give  
n name where the first param is the collection name
```

```
db.shards.save(obj)
```

```
db.shards.stats()
```

```
db.shards.storageSize() - includes free space allocated to this collecti  
on
```

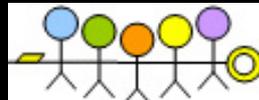
```
db.shards.totalIndexSize() - size in bytes of all the indexes
```

```
db.shards.totalSize() - storage allocated for all data and indexes
```

```
db.shards.update<query, object[, upsert_bool, multi_bool]>
```

```
db.shards.validate() - SLOW
```

```
db.shards.getShardVersion() - only for use with sharding
```



第6章 MongoDB系统管理和维护

本章导读

- 1. 启动和退出
- 2. Web UI
- 3. 集群部署
- 4. 分片
- 5. 异常处理
- 6. 备份和修复
- 7. 安全与认证

6.1 MongoDB 启动和退出

■ 启动服务

```
start mongod --dbpath d:\mongodb\data\config --port 27017 --rest
start mongos --port 30000 --configdb 127.0.0.1:27017
start mongod --dbpath d:\mongodb\data\shard1 --port 10000
start mongod --dbpath d:\mongodb\data\shard2 --port 20000
mongo --port 30000
```

6.1 MongoDB 启动和退出

■ 退出服务

→ctrl + c

→kill -2 PID (SIGINT)

→kill PID (SIGTERM)

→db.shutdownServer()

→禁止使用kill -9 PID (SIGKILL)

[List all commands](#) | [Replica set status](#)

Commands: [buildInfo](#) [cursorInfo](#) [features](#) [listDatabases](#) [serverStatus](#) [top](#)

```
db version v1.8.2-rc2, pdfilec version 4.5
git hash: 373038f53049071fddb5404698c8bcbf99c3b51f
sys info: windows (5, 1, 2600, 2, 'Service Pack 3') BOOST_LIB_VERSION=1_35
uptime: 1269 seconds
```

dbtop (occurrences|percent of elapsed)

write lock % time in write lock, by 4 sec periods

[write locked now](#): false

Log

```
Fri Oct 04 19:06:26 [mongosMain] waiting for connections on port 30000
19:06:26 [websvr] web admin interface listening on port 31000
19:06:27 [Balancer] warning: could not initialize balancer, please check that all shards and config servers are up
19:06:27 [Balancer] will retry to initialize balancer in one minute
19:06:33 [mongosMain] connection accepted from 127.0.0.1:22317 #1
19:07:27 [Balancer] about to contact config servers and shards
19:07:27 [Balancer] config servers and shards contacted successfully
19:07:27 [Balancer] balancer id: microsof-c2f4ea:30000 started at Oct 04 19:07:27
19:07:27 [LockPinger] creating dist lock ping thread for: 127.0.0.1:27017
19:07:27 [Balancer] dist_lock forcefully taking over from: { _id: "balancer", process: "microsof-c2f4ea:1380866660:41", state
19:16:57 .
19:21:57 .
19:26:57 .
```

6.3 MongoDB 集群部署

节点清单

名称	IP	备注
Mongos	127.0.0.1:30000	路由服务器
Configdb	127.0.0.1:27010	配置服务器
Shard1	127.0.0.1:10000	分片服务器1
Shard2	127.0.0.1:20000	分片服务器2
Shard3	127.0.0.1:40000	分片服务器3
...	...	

统一规划，统一配置

6.3 MongoDB 集群部署

■ 配置说明

→ configdb.bat

```
start mongod --dbpath d:\mongodb\data\config --port  
27017 --rest
```

→ mongos

```
start mongos --port 30000 --configdb 127.0.0.1:27017
```

→ shard1

```
start mongod --dbpath d:\mongodb\data\shard1 --port  
10000
```

→ mongo

```
mongo --port 127.0.0.1:30000
```

6.3 MongoDB 集群部署

■ 启动说明-顺序不可逆

Step1-首先启动配置服务器

Step2-启动路由服务器

Step3-启动分片服务器1...2...3...4...

Step4-启动MongoDB客户端

6.4 MongoDB 分片

■ 何时分片

- 机器的磁盘空间不够用
- 单个mongod已经不能够满足写数据的性能需求
- 需要将大数据放到内存中提高性能

6.4 MongoDB 分片

■ 添加分片

- 连接到路由服务器mongos
- 切换到admin数据库use admin(否则无权限错误)
- `db.runCommand({addshard:"127.0.0.1:10000",allowLocal:true})`
- 切换到config数据库use config
- `db.shards.find()`可以查看刚刚创建的shard

```
switched to db config
> db.databases.find()
{ "_id" : "admin", "partitioned" : false, "primary" : "config" }
{ "_id" : "user", "partitioned" : true, "primary" : "shard0000" }
{ "_id" : "book", "partitioned" : false, "primary" : "shard0001" }
{ "_id" : "login", "partitioned" : false, "primary" : "shard0000" }
```

6.4 MongoDB 分片

■ 删除分片

- 连接到路由服务器mongos
- 切换到admin数据库use admin(否则无权限错误)
- db.runCommand({removeshard:"127.0.0.1:10000"})
- 切换到config数据库use config
- db.shards.find()可以查看刚刚创建的shard

```
> use admin
switched to db admin
> db.runCommand(<removeshard:"127.0.0.1:27017">)
{
  "msg" : "draining started successfully",
  "state" : "started",
  "shard" : "shard0002",
  "ok" : 1
}
```

6.3 MongoDB 分片

■ 开启数据分片

```
> use config
switched to db config
> db.databases.find()
{ "_id" : "admin", "partitioned" : false, "primary" : "config" }
{ "_id" : "user", "partitioned" : true, "primary" : "shard0000" }
{ "_id" : "book", "partitioned" : true, "primary" : "shard0001" }
{ "_id" : "login", "partitioned" : false, "primary" : "shard0000" }
> db.databases.find(<enablesharding:"login">)
> db.runCommand(<enablesharding:"login">)
{ "ok" : 0, "errmsg" : "access denied - use admin db" }
> use admin
switched to db admin
> db.runCommand(<enablesharding:"login">)
{ "ok" : 1 }
```

6.4 MongoDB 分片

■ 开启集合分片-片键

```
> use login
switched to db login
> show collecitons
Tue Oct 01 19:25:50 uncaught exception: don't know how to show [collecitons]
> show collections
system.indexes
t_login_info
> db.runCommand(<<"shardcollection":"t_login_info","key":<{"_id":1}>>>)
{ "ok" : 0, "errmsg" : "access denied - use admin db" }
> use admin
switched to db admin
> db.runCommand(<<"shardcollection":"t_login_info","key":<{"_id":1}>>>)
{ "ok" : 0, "errmsg" : "sharding not enabled for db" }
> db.runCommand(<<"shardcollection":"login.t_login_info","key":<{"_id":1}>>>)
{ "collectionsharded" : "login.t_login_info", "ok" : 1 }
```

6.5 MongoDB异常处理

启动失败

```
D:\mongodb\bin>mongod --dbpath d:\mongodb\data\config --port 27017
Fri Oct 04 13:51:22 [initandlisten] MongoDB starting : pid=2988 port=27017 dbpat
h=d:\mongodb\data\config 32-bit

** NOTE: when using MongoDB 32 bit, you are limited to about 2 gigabytes of data

**      see http://blog.mongodb.org/post/137788967/32-bit-limitations
**      with --dur, the limit is lower

Fri Oct 04 13:51:22 [initandlisten] db version v1.8.2-rc2, pdfile version 4.5
Fri Oct 04 13:51:22 [initandlisten] git version: 373038f53049071fddb5404698c8beb
f99e3b51f
Fri Oct 04 13:51:22 [initandlisten] build sys info: windows (5, 1, 2600, 2, 'Ser
vice Pack 3') BOOST_LIB_VERSION=1_35
*****
old lock file: d:\mongodb\data\config\mongod.lock.  probably means unclean shutd
own
recommend removing file and running --repair
see: http://dochub.mongodb.org/core/repair for more information
*****
Fri Oct 04 13:51:22 [initandlisten] exception in initAndListen std::exception: o
ld lock file, terminating
Fri Oct 04 13:51:22 dbexit:
Fri Oct 04 13:51:22 [initandlisten] shutdown: going to close listening sockets..
-
Fri Oct 04 13:51:22 [initandlisten] shutdown: going to flush diaglog...
Fri Oct 04 13:51:22 [initandlisten] shutdown: going to close sockets...
Fri Oct 04 13:51:22 [initandlisten] shutdown: waiting for fs preallocator...
Fri Oct 04 13:51:22 [initandlisten] shutdown: closing all files...
Fri Oct 04 13:51:22 closeAllFiles() finished
Fri Oct 04 13:51:22 dbexit: really exiting now
```

6.5 MongoDB异常处理

■ 执行修复

→ 删除mongod.lock

→ 执行修复: Mongod --repair

```
D:\mongodb\bin>mongod --repair
Fri Oct 04 13:52:14 [initandlisten] MongoDB starting : pid=2672 port=27017 dbpath=/data/db/ 32-bit

** NOTE: when using MongoDB 32 bit, you are limited to about 2 gigabytes of data

**      see http://blog.mongodb.org/post/137788967/32-bit-limitations
**      with --dur, the limit is lower

Fri Oct 04 13:52:14 [initandlisten] db version v1.8.2-rc2, pdf file version 4.5
Fri Oct 04 13:52:14 [initandlisten] git version: 373038f53049071fddb5404698c8bebf99e3b51f
Fri Oct 04 13:52:15 [initandlisten] build sys info: windows (5, 1, 2600, 2, 'Service Pack 3') BOOST_LIB_VERSION=1_35
Fri Oct 04 13:52:15 [initandlisten] exception in initAndListen std::exception: dbpath (/data/db/) does not exist, terminating
Fri Oct 04 13:52:15 dbexit:
Fri Oct 04 13:52:15 [initandlisten] shutdown: going to close listening sockets...
-
Fri Oct 04 13:52:15 [initandlisten] shutdown: going to flush diaglog...
Fri Oct 04 13:52:15 [initandlisten] shutdown: going to close sockets...
Fri Oct 04 13:52:15 [initandlisten] shutdown: waiting for fs preallocator...
Fri Oct 04 13:52:15 [initandlisten] shutdown: closing all files...
Fri Oct 04 13:52:15 closeAllFiles() finished
Fri Oct 04 13:52:15 dbexit: really exiting now
```

6.6 MongoDB备份和恢复

■ 冷备之殇

因为MongoDB所有数据都存放在数据目录下，因此可以采用备份数据目录中所有文件的副本即可实现备份。如此备份的也就是传统的冷备模式，缺点显而易见，一是可能发生数据不一致（备份时间点之后的数据没了）二是备份了一份不可恢复的数据（备份时数据库已经坏了）。

6.6 MongoDB备份和恢复

■ mongodump

MongoDB自带，运行时进行备份的方法，`mongodump`对运行MongoDB进行查询，然后将所有查询文档写入磁盘。因为Mongodump是普通的客户端，所以可以运行mongodb时使用，即便是正在处理其他请求或者执行写入也没问题。

→并非实时快照：备份时正在执行写入的情况

→查询性能干扰：

6.6 MongoDB备份和恢复

■ mongodump

mongodump备份命令可以通过—help查看，下面是一个备份的示例代码：

```
D:\mongodb\bin>mongodump -h 127.0.0.1:30000 -d login -o backup
connected to: 127.0.0.1:30000
DATABASE: login to backup/login
    login.t_login_info to backup/login/t_login_info.bson
        2 objects
    login.system.indexes to backup/login/system.indexes.bson
        1 objects
```

6.6 MongoDB 备份和恢复

■ mongodump



6.6 MongoDB备份和恢复

■ mongodrestore

MongoDB自带的从备份中恢复的工具，通过获取 `mongodump` 的输出结果，并将备份的数据插入到运行的 MongoDB 实例中。示例代码如下：

```
./mongorestore -d foo --drop backup/foo
```

`-d` 支持将备份恢复到与原来不同的数据库中，`--drop` 属性代表在恢复前删除集合（插入操作的局限性）

6.7 MongoDB安全与认证

- 开启安全性检查

```
mongod ... --auth
```