**mongoDB常用命令**

# **安装**

[mongoDB官网](https://www.mongodb.org/)下载安装

# **基础知识**

集合——对应关系数据库中的表

文档——对应关系数据库中的行

# **开启一个客户端访问数据库**

同样的bin文件夹下执行

> mongo –host 172.16.45.3

默认连接至test数据库

# **显示帮助**

> help

# **显示所有数据库名称**

> show dbs

# **切换数据库**

> use test

# **显示当前连接的数据库名称**

> db

# **显示当前数据库所有集合**

> show collections

# **显示数据库支持的方法**

> db.help()

**显示集合支持的方法**

> db.users.help()

# **创建集合**

> db.createCollection("users")

# **插入操作insert**

> db.users.insert({"name":"kiinlam","age":28})

# **查询操作find**

#### **查找所有文档**

> db.users.find()

####查找指定文档

> db.users.find({"name":"kiinlam"})

#### **查询一条**

> db.users.findOne({"name":"kiinlam"})

#### **大于$gt**

> db.users.find({"age":{$gt:22}})

#### **大于等于$gte**

> db.users.find({"age":{$gte:22}})

#### **小于$lt**

> db.users.find({"age":{$lt:22}})

#### **小于等于$lte**

> db.users.find({"age":{$lte:22}})

#### **不等于$ne**

> db.users.find("age":{$ne:22})

#### **或$or**

> db.users.find({$or:[{"name":"kiinlam"},{"name":"cheungkiinlam"}]})

#### **在集合中$in**

> db.users.find("name":{$in:["kiinlam","cheungkiinlam"]})

#### **不在集合中$nin**

> db.users.find("name":{$nin:["kiinlam","cheungkiinlam"]})

#### **正则查询**

> db.users.find({"name":/^k/,"name":/m$/})

#### **筛选查询$where**

// 使用js function作为筛选条件

> db.users.find({$where: function(){return this.name=='kiinlam'}})

#### **限制查询数量limit**

> db.users.find({"age":22}).limit(10)

**更新操作update**

#### **指定文档全部更新，等于覆盖**

> db.users.update({"name":"kiinlam"}, {"name":"cheungkiinlam","age":27})

#### **局部更新一：增量更新$inc**

// age增加2，其他不变

> db.users.update({"name":"kiinlam"}, {$inc:{"age":2}})

#### **局部更新二：字段修改$set**

// age改为20

> db.users.update({"name":"kiinlam"}, {$set:{"age":20}})

#### **新增更新：如果不存在，就新增一条**

// 第三个参数为true

> db.users.update({"name":"kiinlam"}, {$set:{"age":18}}, true)

#### **批量更新**

// 如果匹配多条，默认只改第一条，将第四个参数设为true可全部更新

> db.users.update({"name":"kiinlam"}, {$set:{"age":18}}, true, true)

# **保存操作save**

// 插入新文档，如果不提供"\_id"字段

> db.users.save({"name":"kiinlam", "age":28})

// 更新已存在的文档

> db.users.save({"\_id":"xxx","name":"kiinlam", "age":28})

# **删除操作remove**

删除操作不可恢复

#### **删除所有，但不删除索引**

> db.users.remove({})

#### **删除指定文档**

> db.users.remove({"name":"kiinlam"})

#### **删除一条指定文档，如果有多条结果**

> db.users.remove({"name":"kiinlam"}, true)

完全删除集合，包括索引，应当使用drop

大量删除时，采用复制需要保留的文档到新集合，再用drop删除集合。

# **删除数据库**

> db.dropDatabase()

# **删除集合**

> db.users.drop()

# **计数操作count**

> db.users.count()

> db.users.count({"age":29})

# **唯一值查询distinct**

#### **指定字段有多个相同时，只取一个，返回指定字段的值组合成的数组**

> db.users.distinct("age")

# **分组操作group**

按照age进行分组操作，分组结果存放在user中，值为对应age的name值的数组

key：分组依据

initial：初始化函数，每个不同的age组共享同一个函数

$reduce： 第一个参数为当前文档，第二参数为前一次函数操作的累计对象，第一次为initial对应的对象

> db.users.group({

"key": {"age": true},

"initial": {"user": []},

"$reduce": function(cur,prev){

prev.user.push(cur.name);

}

})

假设有数据如下：

{ "\_id" : ObjectId("55910457607379845607d9e2"), "name" : "kiinlam", "age" : 29 }

{ "\_id" : ObjectId("55910468607379845607d9e3"), "name" : "shadow", "age" : 26 }

{ "\_id" : ObjectId("55910992607379845607d9e5"), "name" : "foo", "age" : 29 }

{ "\_id" : ObjectId("55911fca607379845607d9e6"), "name" : "dd", "age" : 22 }

{ "\_id" : ObjectId("55911fd3607379845607d9e7"), "name" : "mm", "age" : 22 }

{ "\_id" : ObjectId("55911fdf607379845607d9e8"), "name" : "gg", "age" : 22 }

{ "\_id" : ObjectId("55911feb607379845607d9e9"), "name" : "jj", "age" : 22 }

{ "\_id" : ObjectId("55920545ff40738c1fd0a839"), "name" : "zz", "age" : 1 }

分组结果为：

[

{

"age" : 29,

"user" : [

"kiinlam",

"foo"

]

},

{

"age" : 26,

"user" : [

"shadow"

]

},

{

"age" : 22,

"user" : [

"dd",

"mm",

"gg",

"jj"

]

},

{

"age" : 1,

"user" : [

"zz"

]

}

]

#### **更多分组功能**

可选参数: condition 和 finalize。

`condition` —— 过滤条件

`finalize` —— 函数，分组完成后执行

过滤掉age大于22的文档，增加属性标明分组中文档的数量

> db.users.group({

"key": {"age": true},

"initial": {"user": []},

"$reduce": function(cur,prev){

prev.user.push(cur.name);

},

"condition": {"age":{$lte:22}},

"finalize": function(out){

out.count = out.user.length;

}

})

分组结果为：

[

{

"age" : 22,

"user" : [

"dd",

"mm",

"gg",

"jj"

],

"count" : 4

},

{

"age" : 1,

"user" : [

"zz"

],

"count" : 1

}

]

**mapReduce**

map：映射函数，内部调用emit(key,value)，集合按照key进行映射分组。

reduce：简化函数，对map分组后的数据进行分组简化，reduce(key,value)中的key是emit中的key，而value则是emit分组结果的集合。

mapReduce：最后执行的函数，参数为map、reduce和一些可选参数。

> db.users.mapReduce

function ( map , reduce , optionsOrOutString ){

var c = { mapreduce : this.\_shortName , map : map , reduce : reduce };

assert( optionsOrOutString , "need to supply an optionsOrOutString" )

if ( typeof( optionsOrOutString ) == "string" )

c["out"] = optionsOrOutString;

else

Object.extend( c , optionsOrOutString );

var raw = this.\_db.runCommand( c );

if ( ! raw.ok ){

\_\_mrerror\_\_ = raw;

throw Error( "map reduce failed:" + tojson(raw) );

}

return new MapReduceResult( this.\_db , raw );

}

创建map函数

function (){

emit(this.name,{count:1});

}

创建reduce函数

function (key,value){

var result = {count:0};

for(var i = 0; i < value.length; i++){

result.count += value[i].count;

}

return result;

}

执行mapReduce操作

> db.users.mapReduce(map,reduce,{"out":"collection"})

假设有数据如下

{ "\_id" : ObjectId("55910457607379845607d9e2"), "name" : "kiinlam", "age" : 29 }

{ "\_id" : ObjectId("55910468607379845607d9e3"), "name" : "shadow", "age" : 26 }

{ "\_id" : ObjectId("55910992607379845607d9e5"), "name" : "foo", "age" : 29 }

{ "\_id" : ObjectId("55920545ff40738c1fd0a839"), "name" : "zz", "age" : 1 }

{ "\_id" : ObjectId("55911fca607379845607d9e6"), "name" : "foo", "age" : 22 }

{ "\_id" : ObjectId("55911fd3607379845607d9e7"), "name" : "foo", "age" : 22 }

{ "\_id" : ObjectId("55911fdf607379845607d9e8"), "name" : "foo", "age" : 22 }

{ "\_id" : ObjectId("55911feb607379845607d9e9"), "name" : "foo", "age" : 22 }

输出结果

{

"result" : "collection", // 存放最终结果的集合名

"timeMillis" : 28,

"counts" : {

"input" : 8, // 传入文档的次数

"emit" : 8, // emit函数被调用次数

"reduce" : 1, // reduce函数被调用次数

"output" : 4 // 最后返回文档的个数

},

"ok" : 1

}

查看集合collection中的结果

> db.collection.find()

输出结果

{ "\_id" : "foo", "value" : { "count" : 5 } }

{ "\_id" : "kiinlam", "value" : { "count" : 1 } }

{ "\_id" : "shadow", "value" : { "count" : 1 } }

{ "\_id" : "zz", "value" : { "count" : 1 } }

# **游标**

游标只表示一个引用，并不是真正的执行，在需要的时候，通过for循环或next()方法进行遍历读取，枚举结束后，游标销毁，不再返回数据。

申明一个游标

> var list = db.collection.find()

通过forEach遍历游标

> list.forEach(function(i){

print(i.\_id);

})

输出结果

foo

kiinlam

shadow

zz

或者通过next遍历集合

> var list = db.collection.find()

> list.next()

{ "\_id" : "foo", "value" : { "count" : 5 } }

> list.next()

{ "\_id" : "kiinlam", "value" : { "count" : 1 } }

> list.next()

{ "\_id" : "shadow", "value" : { "count" : 1 } }

> list.next()

{ "\_id" : "zz", "value" : { "count" : 1 } }

> list.next()

2015-07-01T11:27:38.186+0800 E QUERY Error: error hasNext: false

at Error (<anonymous>)

at DBQuery.next (src/mongo/shell/query.js:255:15)

at (shell):1:6 at src/mongo/shell/query.js:255

> list

>

# **索引ensureIndex**

#### **建立索引**

// 1为升序，-1为降序

> db.users.ensureIndex({"name":1})

#### **唯一索引**

> db.users.ensureIndex({"name":1},{"unique":true})

#### **组合索引**

> db.users.ensureIndex({"name":1, "age":-1})

#### **查看索引**

> db.users.getIndexes()

#### **按指定索引查询**

> db.users.find({"name":"kiinlam"}).hint({"name":1,"age":1})

#### **删除索引**

// 删除所有自定义索引

> db.users.dropIndexes()

// 删除指定索引

> db.users.dropIndex("name\_1")

# **性能分析函数explain**

> db.users.find().explain("executionStats")

# **主从数据库部署**

#### **创建主数据库master**

> mongod --dbpath=XXX --master

#### **创建从数据库slave**

// 指定从数据库端口--port

// 指定主数据库源--source

> mongod --dbpath=XXX --port=8888 --slave --source=127.0.0.1:27017

#### **后期指定主数据库源**

> mongod --dbpath=XXX --port=8888 --slave

// 后期添加源

// 切换到local数据库

> use local

// 在sources中加入源地址

> db.sources.insert({"host":"127.0.0.1:27017"})

**副本集replSet**

该架构没有特定的主数据库，一个数据库宕机了，另一个数据库会顶上

#### **创建第一个数据库服务器**

// 需要指定集群名及下一个数据库地址

> mongod --dbpath=XXX --port 2222 --replSet mySet/127.0.0.1:3333

#### **创建第二个数据库服务器**

> mongod --dbpath=XXX --port 3333 --replSet mySet/127.0.0.1:2222

#### **初始化副本集**

// 进入任一数据库的admin集合

> mongo 127.0.0.1:2222/admin

// 执行初始化操作

> db.runCommand({

"replSetInitiate":{

"\_id":"mySet",

"members":[

{

"\_id":1,

"host":"127.0.0.1:2222"

},

{

"\_id":2,

"host":"127.0.0.1:3333"

}

]

}

})

#### **仲裁服务器**

// 启动仲裁服务器

> mongod --dbpath=XXX --port 4444 --replSet mySet/127.0.0.1:2222

// 回到admin集合中添加仲裁服务器

> mongo 127.0.0.1:2222/admin

> rs.addArb("127.0.0.1:4444")

// 查看服务器集群状态

> rs.status()

# **分片技术**

将集合进行拆分，将拆分的数据均摊到几个分片上。

主要参与者：

* 客户端
* 路由服务器mongos
* 配置服务器
* 分片数据库实例

#### **开启配置服务器config**

> mongod --dbpath=XXX --port 2222

#### **开启路由服务器mongos**

// 指定配置服务器

> mongos --port 3333 --configdb=127.0.0.1:2222

#### **开启分片数据库服务器mongod**

> mongod --dbpath=XXX --port 4444

> mongod --dbpath=XXX --port 5555

#### **服务配置**

// 进入mongos数据库admin集合

> mongo 127.0.0.1:3333/admin

// 添加分片服务器addshard

> db.runCommand({

"addshard":"127.0.0.1:4444",

"allowLocal":true

})

> db.runCommand({

"addshard":"127.0.0.1:5555",

"allowLocal":true

})

// 开启数据库test的分片功能enablesharding

> db.runCommand({"enablesharding":"test"})

// 指定集合中分片的片键users.name

> db.runCommand({"shardcollection":"test.users","key":{"name":1}})

// 在mongos中查看数据分片情况

> use test

> db.printShardingStatus()

# **运维**

运维通常会涉及到以下4个方面

* 安装部署
* 状态监控
* 安全认证
* 备份和恢复

#### **安装部署为windows服务**

// 指定日志路径，添加install参数

> mongod --dbpath=XXX --logpath=XXX --port=2222 --install

// 启动服务

> net start MongoDB

#### **状态监控**

###### **静态统计**

*db.stats()*

// 查看单个数据库状态

> db.stats()

stats比较简单，可以参考[db.stats()](http://www.cnblogs.com/xuegang/archive/2011/10/13/2209965.html)一文

*db.serverStatus()*

// 查看整个mongodb的状态

// 进入admin集合

> mongo 127.0.0.1:2222/admin

// 查看状态

> db.serverStatus()

serverStatus的参数很多，可以参考[db.serverStatus()](http://www.cnblogs.com/xuegang/archive/2011/10/13/2210339.html)一文

###### **实时统计**

> mongostat --port 2222

#### **安全认证**

*TODO*

有点复杂，偷懒了，参考[安全认证](http://docs.mongodb.org/manual/security/)

#### **备份和恢复**

// 备份test数据库到D:\mongodb\backup

> mongodump --port 2222 -d test -o D:\mongodb\backup

// 恢复数据，drop表示恢复前删除原有数据

> mongorestore --port 2222 -d test --drop D:\mongodb\backup