MongoDB 学习

2012.07.23 谭志军 @人人

目 录

1.MongoDB 特性

1.1. 面向文档 Document-oriented

MongoDB 是面向文档，非关系型数据库。基本的思路是将"row"的概念换成更加灵活的"document"模型。面向文档的方式可以将文档或者数组内嵌起来，用一条记录就能处理复杂的层次关系。

1.2. 高可用性 High availability

Replica sets 集群中，master 自动故障转移

1.3. 易扩展 Easy scalability

随着应用数据集的大小飞速增加，T 级别数据变的司空见惯；使用传统的关系型数据库（eg: MySQL）的时候，数据拆分成了日常工作之一。MongoDB 正是在这样的环境中降生，他的设计者们最初就考虑到扩展的问题。他所采用的面对文档的数据模型使其可以自动在多台服务器之间分割数据，并且自动平衡集群的数据和负载。

1.4. 丰富的查询语言 Rich query language

1.5. 模式自由 Schema free

一个 collection 中的 document 可以是多种多样的，但不建议使用(管理困难，查询效率低下，不方便索引)。

2.MongoDB 基本组成

2.1. document --> row

多个键 Key 及其关联的值 Value 有序的放置在一起便是文档。

{"Company"：＂renren.com＂，＂xyz＂：3}

a、区分类型、大小写

b、键 Key 不能含有\0 ；"." 和"$"有特别意义；"_" 系统保留

2.2. collection --> table

a、命名不能用空字符串""、\0、"$"；"system." 系统保留

2.3. db

a、命名不能用空字符串""、空格' '、.、$、/、\；应全部小写；最多 64 字节

b、3 个特殊 db

admin "root"数据库，这个 db 下的用户集成所有数据库的权限，shutdown 等特定服务器端命令只能从 admin 下运行。

local 不会被复制

config 保存分片相关信息

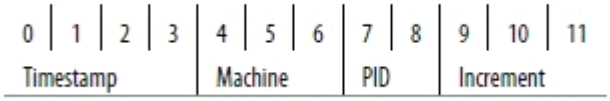
2.4. 数据类型

数字、时间、字符串、ObjectId()、null、bool、数组、内嵌文档

ObjectId "_id" 默认类型，使用 12 字节的存储空间，每个字节两位十六进制数字，是

一个 24 位的字符串。

ObjectId("5016389ca188cf62e321b6d8")



自定义自增 _id
//初始化
db.ids.save({'_id':'main','id':0});
//每次使用的时候自增 1
db.ids.update({"_id":"main"},{$inc:{'id':1}});
//取得当前 ID 值，insert 目标 collection

3.Creating，Updating，and Deleting Documents

3.1. INSERT

```
mongos> db.user.insert({"name":"RRDBA","online":{"email":"renren.db@renre
n-inc.com","renren":"DBA2511","xiaonei":"dba2511","kaixin":"dba2511"}})
mongos> db.user.save({"_id":1, "name":"rrdba"}) //save->replace
```

3.2. REMOVE

删除整个文档内容，但不会删除 collection 本身和索引

```
mongos> db.user.remove()
```

删除匹配的 documents

```
mongos> db.user.remove({"name":"rrdba"})
```

3.3. UPDATE

db.collection.update(criteria, objNew, upsert[false], multi[false])

- *criteria* - query which selects the record to update;
- *objNew* - updated object or $ operators (e.g., $inc) which manipulate the object
- *upsert* - if this should be an "upsert" operation; that is, if the record(s) do not exist, insert one. **Upsert only inserts a single document**.
- *multi* - indicates if all documents matching *criteria* should be updated rather than just one. Can be useful with the $ operators below.

```
mongos> db.user.update({"name":"RRDBA"},{"$set":{"age":8}})

mongos>  db.user.update({"name":"RRDBA"},{"$set":{"online.renren":"DB"}})
```

```
mongos> db.user.findOne()
{
        "_id" : ObjectId("501787d1a6b8a79001128de0"),
        "age" : 8,
        "name" : "RRDBA",
        "online" : {
                "email" : "renren.db@renren-inc.com",
                "kaixin" : "dba2511",
                "renren" : "DB",
                "xiaonei" : "dba2511"
        }
}
```

$upsert：如果不存在"age":15 的记录，则写入一条

```
mongos> db.user.update({"age":15},{"$set":{"age":15}},true)
```

$unset: 去除"tel"和"online.renren" '列'

```
mongos>  db.user.update({"name":"RRDBA"},{"$unset":{"tel":1}})

mongos>  db.user.update({"name":"RRDBA"},{"$unset":{"online.renren":1}})
```

$inc: "age" 键的值增加 3

```
mongos> db.user.update({"name":"RRDBA"},{"$inc":{"age":3}})
```

$rename: (1.7.2+) 修改键名

```
mongos> db.user.update({},{"$rename":{"online":"onl"}})
```

_id 不允许 update

```
mongos> db.user.update({""_id" : ObjectId("5019f269eef1bfda39808927")},{$
set:{"_id":1}})
Mod on _id not allowed
```

3.4. 数组操作

$push: 数组 email 添加新元素

```
mongos> db.user.update({"name":"RRDBA"},{"$push":{"email":"dba@xiaonei.
com"}})
```

$addToSet: 存在则跳过，不存在就添加； **注：**upsert 对数组无效

```
mongos> db.user.update({"name":"RRDBA"},
... {"$addToSet":{"email":"dba@xiaonei.com"}})
```

$addToSet..$each: 数组 email 添加多个元素

```
mongos> db.user.update({"name":"RRDBA"},{"$addToSet":
... {"email":{"$each":["dba@xiaonei.com","xyz@kaixin.com"]}}})
mongos> db.user.update({},{"$pushAll":
... {"email":["dba@kaixin.com","dba@renren.com"]}})
```

```
mongos> db.user.findOne()
{
        "_id" : ObjectId("50179a2f85a451596b437b15"),
```

```
        "age" : 10,
        "email" : [
                "dba@kaixin.com",
                "dba@xiaonei.com",
                "xyz@kaixin.com"
        ],
        "name" : "RRDBA",
        "online" : {
                "kaixin" : "dba2511",
                "xiaonei" : "dba2511"
        }
}
```

$pop: 删除数组 email 最后一个元素

```
mongos> db.user.update({},{"$pop":{"email":1}})
```

$pop: 删除数组 email 第一个元素

```
mongos> db.user.update({},{"$pop":{"email":-1}})
```

$pull: 删除数组 email 元素"x@kaixin.com" $pullAll ~ $pushAll

```
mongos> db.user.update({},{"$pull":{"email":"x@kaixin.com"}})
```

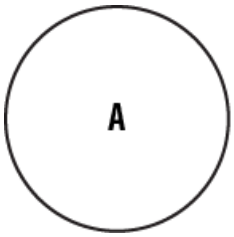
0 1..$: 数组定位修改

```
mongos> db.user.update({"name":"RRDBA"},{"$set":{"email.0":"@COM"}})
mongos> db.user.update({"email":"@COM"},{"$set":{"email.$":"db@xiaonei.com"}})
```

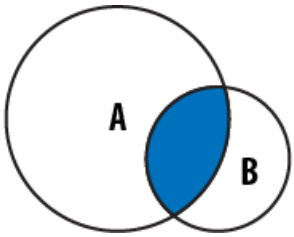
4.Query -- find()

```
mongos> db.user.find({"name":"RRDBA"},{"name":1,"_id":0})
{ "name" : "RRDBA" }
{ "name" : "RRDBA" }
```

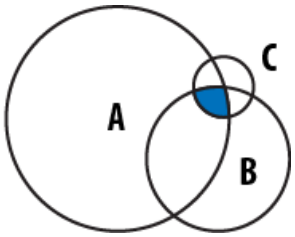
4.1. and 查询,将最严格的条件放最前



A(40000 documents)



B(9000 documents)



C(200 documents)

如果我们从条件 C 开始，再 B，再 C；则只扫描了 200documents 。

4.2. 条件操作符:

数字、时间
$lt <
$lte <=
$gt >
$gte >=

不限数据类型
$ne =
$in
$nin

```
mongos> db.user.find({"name":{"$ne":"RRDBA"}})

mongos> db.user.find({"name":{"$in":["RRDBA","MySQL","DB"]}})

mongos> db.user.find({"name":{"$nin":["RRDBA","MySQL","DB"]}})
```

$or：使用"$or"查询，用来完成多个键值的任意给定值

```
mongos>  db.user.find({"$or":[{"name":"RRDBA"},{"age":{"$in":[21,22]}}]})
```

4.3. $not 元条件句，可以用在任何其他条件之上

```
mongos> db.user.find({"age":{"$mod":[5,2]}})     //$mod 取模

mongos> db.user.find({"age":{"$not":{"$mod":[5,2]}}})
```

4.4. null null 不仅匹配自身为空，而且匹配“不存在”

```
mongos> db.user.find({"name":{"$in":[null]}})
{ "_id" : ObjectId("5017ab3eba2013555620993f"), "age" : 21 }
{ "_id" : ObjectId("5017aba2ba20135556209940"), "age" : 22 }
{ "_id" : ObjectId("5017b8a3ba20135556209941"), "age" : 24 }
{ "_id" : ObjectId("5019069c71c9800ed90b8237"), "name" : null }

mongos> db.user.find({"name":{"$in":[null],"$exists":true}})
{ "_id" : ObjectId("5019069c71c9800ed90b8237"), "name" : null }
```

4.5. 正则

MongoDB 使用 Perl 兼容的正则表达式（PCRE），支持前缀索引

```
mongos> db.user.find({"name":/RR/})

mongos> db.user.find({"name":/^RR/})

mongos> db.user.find({"name":/Rr/i})
```

4.6. javascript 表达式和 $where

```
{ "_id" : 0, "x" : 1, "y" : 2, "z" : 0, "date" : ISODate("2012-07-11T10:15:23.631Z"), "name" : "test" }
{ "_id" : 1, "x" : 1, "y" : 2, "z" : 1, "date" : ISODate("2012-07-11T10:15:23.631Z"), "name" : "test" }
{ "_id" : 2, "x" : 1, "y" : 2, "z" : 2, "date" : ISODate("2012-07-11T10:15:23.631Z"), "name" : "test" }
{ "_id" : 3, "x" : 1, "y" : 2, "z" : 3, "date" : ISODate("2012-07-11T10:15:23.631Z"), "name" : "test" }
```

如上需查询 x + z = 3 的文档，只能用"$where"子句借助 javascript 来完成，以下语

法等价：

```
PRIMARY> db.t2.find({$where:"this.x + this.z == 3"})
PRIMARY> db.t2.find("this.x + this.z == 3")
PRIMARY> f = function() { return this.x + this.z == 3;} ; db.t2.find(f) ;
```

注意：不是非常必要，一定要避免使用"$where"查询，因为他们在速度上比常规查询慢很多。每个文档都要从 BSON 转化成 JavaScript 对象，然后通过"$where"的表达式来运行。同样还不能利用索引。

4.7. 游标

http://www.mongodb.org/display/DOCS/Queries+and+Cursors

```
for( var c = db.parts.find(); c.hasNext(); ) {
    print( c.next());
}
db.users.find().forEach( function(u) { print("user: " + u.name); } );
```

4.8. 数组查询

a. 数组元素查询

b. 数组元素精准匹配查询，对元素顺序有要求

c. 数组定位查询

```
mongos> db.u.find()
```

```
{ "_id" : 1, "email" : [ "db@xiaonei.com", "DBA@kaixin.com", "xyz@kaixin.com" ] }
{ "_id" : 2, "email" : [ "db2@xiaonei.com", "dba@kaixin.com", "xyz@kaixin.com" ] }
{ "_id" : 3, "email" : [ "db3@xiaonei.com", "dba@kaixin.com", "xyz@kaixin.com" ] }

mongos> db.u.find({"email":"dba@kaixin.com"})

{ "_id" : 2, "email" : [ "db2@xiaonei.com", "dba@kaixin.com", "xyz@kaixin.com" ] }
{ "_id" : 3, "email" : [ "db3@xiaonei.com", "dba@kaixin.com", "xyz@kaixin.com" ] }

mongos> db.u.find({"email":["db@xiaonei.com","DBA@kaixin.com","xyz@kaix
in.com" ]})

{ "_id" : 1, "email" : [ "db@xiaonei.com", "DBA@kaixin.com", "xyz@kaixin.com" ] }

mongos> db.u.find({"email":["db@xiaonei.com","xyz@kaixin.com","DBA@kaix
in.com"]})

mongos> db.u.find({"email.1":"dba@kaixin.com"})

{ "_id" : 2, "email" : [ "db2@xiaonei.com", "dba@kaixin.com", "xyz@kaixin.com" ] }
{ "_id" : 3, "email" : [ "db3@xiaonei.com", "dba@kaixin.com", "xyz@kaixin.com" ] }
```

$all 查询数组元素既有"db@xiaonei.com" 又有 "DBA@kaixin.com" 的 documents，与顺序无关

```
mongos> db.u.find({"email":{$all:["db@xiaonei.com","DBA@kaixin.com"]}})
{ "_id" : 1, "email" : [ "db@xiaonei.com", "DBA@kaixin.com", "xyz@kaixin.com" ] }
```

$size 查询包含 3 个元素数组

```
mongos> db.u.find({"email":{$size:3}})
```

$slice 返回数组的子集合。 eg: 返回 weibo 前 10 条评论

```
mongos> db.posts.findOne({},{"coms":{"$slice":10}})  //first 2

mongos> db.posts.findOne({},{"coms":{"$slice":-10}})  //last 2

mongos> db.posts.findOne({},{"coms":{"$slice":[-20,10]}})

                                                    //20 from end,limit 10
```

5.索引

MongoDB 索引用法与 MySQL 基本相同。

5.1. 示例：添加索引名称为 i_n 的唯一索引，且去掉重复

```
db.user.ensureIndex({"name":1},{"unique":true,"dropDups":true},{"name":"i_n"})
```

5.2. 地理空间索引

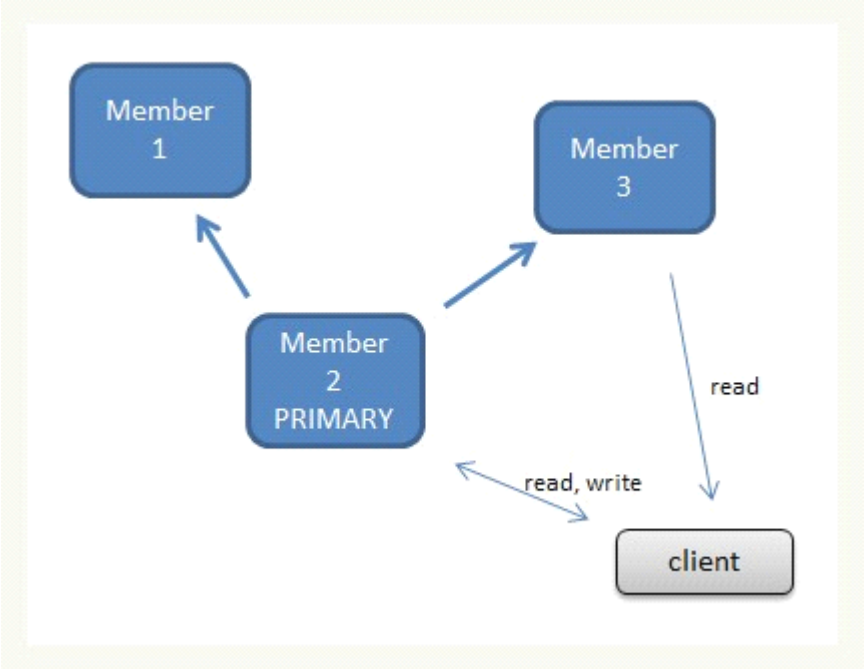
"gps" 键值必须是某种形式的一对值：一个包含两个元素的数组或者包含两个键的内嵌文档：

```
{ "gps" : [ 0, 100 ] }
{ "gps" : { "x" : -30, "y" : 60 } }
{ "gps" : { "latitude" : -180, "longitude" : 180 } }
```

```
mongos> for(i=-180;i<=179;++i) {for(j=-180;j<=179;++j) { db.map.insert({"gps":[i,j]}) }; }
mongos> db.map.ensureIndex({"gps":"2d"})
mongos> db.map.find({"gps":{$near:[126,10]}}).limit(10)
// 返回距离[126,10]最近的 10[默认 100]个文档，由近及远。
mongos> db.runCommand({geoNear:"map",near:[126,10],$maxDistance:3,num:10});
```

```
// geoNear 还会返回每个文档到查询点的距离
mongos> db.map.find({"gps":{$within:{$box:[[126,10],[128,11]]}}})
// $within..$box 指定矩形查询，第一个元素左下角坐标，第二个元素右上角坐标
mongos> db.map.find({"gps":{$within:{$center:[[126,10],2]}}})
// $within..$center 指定圆心，半径查询圆形内所有文档
```

6. Replica sets



Replica sets 集群最大特点是自动故障转移。若 primary 节点出现问题，会将其中 secondary 节点会提升为 primary ，确保对外服务正常。

MongoDB 操作日志 oplog (operation log)，类似 MySQL 的 binary log，作为保持

数据同步的机制。oplog 存放在特殊的数据库 local 中。默认情况下，64 位 mongodb 会初始化 5%的硬盘空间来存放 oplog ，可通过 oplogSize 参数设定。

```
PRIMARY> use local
switched to db local
PRIMARY> db.oplog.rs.findOne({"op":"i"})
{
        "ts" : {
                "t" : 1344566641000,
                "i" : 1
        },
        "h" : NumberLong("-1453879365436008595"),
        "op" : "i",
        "ns" : "test.y",
        "o" : {
                "_id" : ObjectId("50247571060baf3627f4a44c"),
                "save" : 1
        }
}
```

ts：操作时间戳　op：操作类型　ns：collection　o：执行操作的文档内容

Replica sets 集群设置

1、启动 mongod 进程

```
# bin/mongod --dbpath="./data1" --fork --logpath="./logs/m1.log" --port 7701 --replSet rr_set
# bin/mongod --dbpath="./data1" --fork --logpath="./logs/m1.log" --port 7702 --replSet rr_set
# bin/mongod --dbpath="./data1" --fork --logpath="./logs/m1.log" --port 7703 --replSet rr_set
```

2、初始化

```
# bin/mongo 127.0.0.1:7701/admin
> rs.initiate()
{
    "info2" : "no configuration explicitly specified -- making one",
    "info" : "Config now saved locally.  Should come online in about a minute.",
    "ok" : 1
}
```

3、添加新节点

```
PRIMARY> rs.add("host":"mongo168.com:7702")
{ "ok" : 1 }
```

4、查看状态

```
PRIMARY> rs.status()
{
        "set" : "rr_set",
        "date" : ISODate("2012-08-15T10:51:55Z"),
        "myState" : 1,
        "syncingTo" : "mongo168.com::7701",
        "members" : [
                {
                        "_id" : 0,
                        "name" : "mongo168.com:7701",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "PRIMARY",
```

```
                        "uptime" : 17921,
                        "optime" : {
                                "t" : 1345005101000,
                                "i" : 1
                        },
                        "optimeDate" : ISODate("2012-08-15T04:31:41Z"),
                        "lastHeartbeat" : ISODate("2012-08-15T10:51:55Z"),
                        "pingMs" : 0
                },
                ... ... ...
        ],
        "ok" : 1
}
PRIMARY> db.printReplicationInfo()
PRIMARY> db.printSlaveReplicationInfo()
source:    mongo168.com:7702
           syncedTo: Tue Aug 07 2012 09:28:10 GMT+0800 (CST)
                     = 0 secs ago (0hrs)
source:    mongo168.com:7703
           syncedTo: Tue Aug 07 2012 09:28:10 GMT+0800 (CST)
                     = 0 secs ago (0hrs)
```