MongoDB

1）MongoDB介绍

1.1）什么是MongoDB?

- MongoDB是由C++编写的，基于分布式文件存储的开源数据库系统，将数据存储为一个文档，数据结构由键值对组成。MongoDB文档类似于JSON对象，属于NoSQL中的一员。

1.2）主要特点

- MongoDB 是一个面向文档存储的数据库，操作起来比较简单和容易。
- 可以在MongoDB记录中设置任何属性的索引 (如：FirstName="Sameer",Address="8 Gandhi Road")来实现更快的排序。
- 可以通过本地或者网络创建数据镜像，这使得MongoDB有更强的扩展性。
- 如果负载增加（需要更多的存储空间和更强的处理能力），它可以分布在计算机网络中的其他节点上，这就是所谓的分片。

1.3）MongoDB下载使用（Linux）

```
# 下载MongoDB所需要的依赖包
sudo yum install -y libcurl openssl

# 下载MongoDB tgz包，并进行解压
wget https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-rhel70-5.0.3.tgz
tar -zxvf mongodb-linux-x86_64-rhel70-5.0.3.tgz

# 配置系统变量 /etc/profile
# export PATH=/opt/module/mongoDB/bin:$PATH
source /etc/profile

# 创建数据文件夹和日志文件夹，进行指明，启动后默认端口为27017，默认只可本地访问
mongod --dbpath /opt/module/mongodb/data --logpath
/opt/module/mongodb/log/mongod.log --fork

# 启动mongodb
./mongo
```

1.4）MongoDB配置登录验证与创建配置文件进行启动

- 创建新用户、赋予权限

```
> use admin
switched to db admin
> db.createUser({user:"admin",pwd:"password",roles:["root"]})
Successfully added user: { "user" : "admin", "roles" : [ "root" ] }
> db.auth("admin", "password")
1us
```

- 新建配置文件mongodb.conf，以后使用配置进行启动

```
#数据文件存放目录
dbpath = /opt/module/mongodb/data/db
#日志文件存放目录
logpath = /opt/module/mongodb/logs/mongodb.log
#端口
port = 27017
#以守护进程的方式启用，即后台运行;默认false
fork = true
# 关闭web管理访问，默认关闭27018端口访问，这个是在prot端口上加1000
#httpinterface = true
#是否开启权限验证
auth = true
#绑定ip，让其能够通过外网访问， 0.0.0.0代表所有
bind_ip = 0.0.0.0
```

- 以配置文件中的配置进行启动MongoDB服务

```
./mongod -f mongodb.conf
```

- MongoDB的url连接

```
mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:portN]]]
[/[database][?options]]
```

2）MongoDB术语介绍

SQL术语/概念	MongoDB术语/概念	解释/说明
database	database	数据库
table	collection	数据库表/集合
row	document	数据记录行/文档
column	field	数据字段/域
index	index	索引
table joins		表连接,MongoDB不支持
primary key	primary key	主键,MongoDB自动将_id字段设置为主键

2.1）数据库

- 基本命令：show dbs查看所有数据库，use database切换数据库（没有则新建），db查看当前数据库
- 保留数据库：
  - **admin**： 从权限的角度来看，这是"root"数据库。要是将一个用户添加到这个数据库，这个用户自动继承所有数据库的权限。一些特定的服务器端命令也只能从这个数据库运行，比如列出所有的数据库或者关闭服务器。
  - **local:** 这个数据永远不会被复制，可以用来存储限于本地单台服务器的任意集合

- **config**: 当Mongo用于分片设置时，config数据库在内部使用，用于保存分片的相关信息。

2.2）文档

- 文档是一组键值(key-value)对(即 BSON)。MongoDB 的文档不需要设置相同的字段，并且相同的字段不需要相同的数据类型，这与关系型数据库有很大的区别，也是 MongoDB 非常突出的特点。
- 需要注意的是：
  1. 文档中的键/值对是有序的。
  2. 文档中的值不仅可以是在双引号里面的字符串，还可以是其他几种数据类型（甚至可以是整个嵌入的文档)。
  3. MongoDB区分类型和大小写。
  4. MongoDB的文档不能有重复的键。
  5. 文档的键是字符串。除了少数例外情况，键可以使用任意UTF-8字符。
- 文档键命名规范：
  1. 键不能含有\0 (空字符)。这个字符用来表示键的结尾。
  2. “.”和“$”有特别的意义，只有在特定环境下才能使用。
  3. 以下划线"_"开头的键是保留的(不是严格要求的)。

2.3）集合

- 集合就是 MongoDB 文档组（相当于RDBMS中的表，集合没有固定的结构，当第一个文档插入时，集合就会被创建。
- 合法的集合名：
  1. 集合名不能是空字符串""。
  2. 集合名不能含有\0字符（空字符)，这个字符表示集合名的结尾。
  3. 集合名不能以"system."开头，这是为系统集合保留的前缀。
  4. 用户创建的集合名字不能含有保留字符。有些驱动程序的确支持在集合名里面包含，这是因为某些系统生成的集合中包含该字符。除非你要访问这种系统创建的集合，否则千万不要在名字里出现$。
- 固定大小的集合**Capped Collection**，有很高的性能以及队列过期的特性（过期按照插入的顺序）。使用命令`db.createCollection("mycoll", {capped:true, size:100000})`。特性如下：
  1. 在 capped collection 中，你能添加新的对象。
  2. 能进行更新，然而，对象不会增加存储空间。如果增加，更新就会失败 。
  3. 使用 Capped Collection 不能删除一个文档，可以使用 drop() 方法删除 collection 所有的行。
  4. 删除之后，你必须显式的重新创建这个 collection。
  5. 在32bit机器中，capped collection 最大存储为 1e9(1X109)个字节。

2.5）元数据

- 数据库的信息是存储在集合中，它们使用了系统的命名空间

集合命名空间	描述
dbname.system.namespaces	列出所有名字空间。
dbname.system.indexes	列出所有索引。
dbname.system.profile	包含数据库概要(profile)信息。
dbname.system.users	列出所有可访问数据库的用户。
dbname.local.sources	包含复制对端（slave）的服务器信息和状态。

- 在{{system.indexes}}插入数据，可以创建索引。但除此之外该表信息是不可变的(特殊的drop index命令将自动更新相关信息)。

  {{system.users}}是可修改的。 {{system.profile}}是可删除的。

2.6）MongoDB的数据类型

- 常用类型：String、Integer、Boolean、Double、**Min/Max keys**（将一个值与 BSON（二进制的 JSON）元素的最低值和最高值相对比）、Array、**Timestamp**（记录文档修改或添加的具体时间）、Object（内嵌文档）、Null、Symbol、Date、**Object ID**（对象ID，用于创建文件）、Binary Data（二进制数据）、Code（文档中存储的JS代码）、Regular Expression（正则表达式）。
- 重要的数据类型：
  - Object ID：由12个字节组成，前4个表示创建的unix时间戳，接下来3个字节表示机器标识码，后2个字节由进程id组成PID、最后3个字节为随机数。由于ObjectID已经包含了创建时间戳，因此无需再保存此元素，可通过ObjectID的getTimestamp方式获取；通过ObjectID的str展示值。
  - String：BSON的字符串为UTF-8编码。
  - Date：表示当前距离 Unix新纪元（1970年1月1日）的毫秒数。日期类型是有符号的, 负数表示 1970 年之前的日期。

3）MongoDB的使用

3.1）数据库的使用

```
# 新建数据库，存在则切换至该数据库，不存在则新建（只有当存入第一个文档时才真正新建）
> use testdb
switched to db testdb

# 查看当前数据库
> db
testdb

# 查看所有数据库
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB

# 删除数据库
```

```
> db.dropDatabase()
{ "ok" : 1 }
```

3.2）集合的使用

- 创建集合的可选参数：

字段	类型	描述
capped	布尔	为 true，则创建固定集合。当大小达到最大值时，它会自动覆盖最早的文档。**当该值为 true 时，必须指定 size 参数。**
size	数值	为固定集合的最大值，为字节数。（优先于max）
max	数值	指定固定集合中包含文档的最大数量。

```
# 新建集合
> db.createCollection("testCollection", {capped:true, size:100, max:100})
{ "ok" : 1 }

# 查看当前数据库下所有集合，两个指令都可以
> show collections
testCollection
> show tables
testCollection

# 删除集合
> db.testCollection.drop()
true
```

3.3）文档的使用

3.3.1）文档的增加

- **save()**与**insert()**命令可插入新文档。但是当指定ObjectID即主键时，若主键存在，则save相当于更新，而insert则会报错，抛 org.springframework.dao.DuplicateKeyException 异常，提示主键重复。（若插入的集合不存在，则会自动创建该集合）

```
> db.testcollection.save({name:"JIA",age:23})
WriteResult({ "nInserted" : 1 })

> db.testcollection.save({ "_id" : ObjectId("617f994fc8d545dc9dcfc9c9"), "name"
: "JIA", "age" : 24 })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.testcollection.insert({ "_id" : ObjectId("617f994fc8d545dc9dcfc9c9"),
"name" : "JIA", "age" : 24 })
WriteResult({
        "nInserted" : 0,
        "writeError" : {
                "code" : 11000,
```

```
                "errmsg" : "E11000 duplicate key error collection:
testdb.testcollection index: _id_ dup key: { _id:
ObjectId('617f994fc8d545dc9dcfc9c9') }"
        }
})
```

- **insertOne()** 和**insertMany()**可用于插入一个或多个文档

```
> var document = db.testcollection.insertOne({name:"ZURI"})
> document
{
        "acknowledged" : true,
        "insertedId" : ObjectId("617f9acbc8d545dc9dcfc9ca")
}

> var document = db.testcollection.insertMany([{name:"JIA"}, {name:"ZURI"}])
> document
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("617f9b0ec8d545dc9dcfc9cb"),
                ObjectId("617f9b0ec8d545dc9dcfc9cc")
        ]
}
```

- 一次性插入多个文档

```
> var arr = [];
> for (var i = 1;i < 10;i++){
... arr.push({num : i});
... }
9
> db.testcollection.insert(arr)
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 9,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
```

3.3.2）文档的更新

- **save()**和**update()**都可用于文档的更新，但save是根据主键ID进行整个替换，而update则可以进行部分字段的更新。db.colletion.update({query}, {update}, {[][][upsert], [multi], [writeConcern]})的参数说明如下（**书写可选项时不一定要写出可选项名称，会按照顺序自动匹配**）：
    - query：update的查询条件，类型于sql的where
    - update：修改操作
    - upsert：布尔值，为true表示不存在update的记录则新建，默认为false

- multi：布尔值，为true表示是否修改根据条件查到的所有文档，false表示只修改找到的第一个文档，默认为false
- writeConcern：抛出异常的级别

```
> db.testcollection.update({'name':'JIA'}, {$set:{'name':'ZURI'}}, {multi:true})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

- **updateOne()**和**updateMany()**用于更新一个或多个文档

```
> db.testcollection.updateOne({name:"abc"}, {$set:{age:28}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

> db.testcollection.updateMany({age:{$gt:"10"}}, {$set:{status:"xyz"}})
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
```

- 操作符：
  - $set：若该键值对不存在则新建，存在则修改
  - $unset：若该字段存在则移除
  - $setOrInsert：query查询到数据存在时不进行操作
  - $inc：数值类增加

```
# unset后面的字段才重要，值是无所谓的
> db.testcollection.update({name:"ZURI"}, {$unset:{age:21}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

# 因为能查到该记录，所以不操作
> db.testcollection.update({name:"ZURI"}, {$setOnInsert:{age:21}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
```

- WriteConcern可选值：
  - WriteConcern.NONE:没有异常抛出
  - WriteConcern.NORMAL:仅抛出网络错误异常，没有服务器错误异常
  - WriteConcern.SAFE:抛出网络错误异常、服务器错误异常；并等待服务器完成写操作。
  - WriteConcern.MAJORITY: 抛出网络错误异常、服务器错误异常；并等待一个主服务器完成写操作。
  - WriteConcern.FSYNC_SAFE: 抛出网络错误异常、服务器错误异常；写操作等待服务器将数据刷新到磁盘。
  - WriteConcern.JOURNAL_SAFE:抛出网络错误异常、服务器错误异常；写操作等待服务器提交到磁盘的日志文件。
  - WriteConcern.REPLICAS_SAFE:抛出网络错误异常、服务器错误异常；等待至少2台服务器完成写操作。

3.3.3）文档的删除

- ~~remove()~~用于文档的删除。db.collection.remove({query}, {[justOne], {writeConcern}})的参数说明如下：query（可选）、justOne（可选，true或者1表示只删除一个）。

```
> db.testcollection.remove({name:'ZURI'})
WriteResult({ "nRemoved" : 1 })
```

- **deleteMany()**和**deleteOne()**用于最新版本的删除文档，后面接查询参数（空为{}，不可省略）

```
> db.testcollection.deleteOne({name:'abc'})
{ "acknowledged" : true, "deletedCount" : 1 }
> db.testcollection.deleteMany({status:'xyz'})
{ "acknowledged" : true, "deletedCount" : 2 }
> db.testcollection.deleteMany({})
{ "acknowledged" : true, "deletedCount" : 0 }
```

3.3.4）文档的查询

- **find(query, projection)**用于文档的查询，query与projection均为可选，后者表示需要显示的键，find()后面接pretty()用于格式化显示文档。同时，还有**findOne()**指令只展示一条。

```
# findOne()后面不可以接pretty()
> db.testcollection.findOne({name:'JIA'})
{ "_id" : ObjectId("617fa8afc8d545dc9dcfc9db"), "name" : "JIA" }

> db.testcollection.find().pretty()
{ "_id" : ObjectId("617fa8afc8d545dc9dcfc9db"), "name" : "JIA" }
{ "_id" : ObjectId("617fa8c2c8d545dc9dcfc9dc"), "name" : "JIA123" }
```

- MongoDB的"where"操作符

操作	格式
等于	`{<key>:<value>}`
小于	`{<key>:{$lt:<value>}}`
小于或等于	`{<key>:{$lte:<value>}}`
大于	`{<key>:{$gt:<value>}}`
大于或等于	`{<key>:{$gte:<value>}}`
不等于	`{<key>:{$ne:<value>}}`
包含	`{<key>:/<value>/`
开头	`{<key>:/^<value>/}`
结尾	`{<key>:/<value>$/}`

```
# 大于50小于80的写法
db.testcollection.find({age: {$gt:50, $lt:80}})

# 包含，开头，结尾
> db.testcollection.find({name:/I/})
{ "_id" : ObjectId("617fa8afc8d545dc9dcfc9db"), "name" : "JIA", "age" : 21 }
{ "_id" : ObjectId("617fa8c2c8d545dc9dcfc9dc"), "name" : "JIA123" }

> db.testcollection.find({name:/^J/})
{ "_id" : ObjectId("617fa8afc8d545dc9dcfc9db"), "name" : "JIA", "age" : 21 }
{ "_id" : ObjectId("617fa8c2c8d545dc9dcfc9dc"), "name" : "JIA123" }


> db.testcollection.find({name:/3$/})
```

```
{ "_id" : ObjectId("617fa8c2c8d545dc9dcfc9dc"), "name" : "JIA123" }
```

- MongoDB的and（，）与or（$or）

```
> db.testcollection.find({"name":"JIA", "age":21})
{ "_id" : ObjectId("617fa8afc8d545dc9dcfc9db"), "name" : "JIA", "age" : 21 }

> db.testcollection.find({$or: [{name:"JIA123"}, {age : 21}]})
{ "_id" : ObjectId("617fa8afc8d545dc9dcfc9db"), "name" : "JIA", "age" : 21 }
{ "_id" : ObjectId("617fa8c2c8d545dc9dcfc9dc"), "name" : "JIA123" }
```

- **projection**参数，指定返回的键，后面接0/1（exclusion/inclusion），0表示不返回，1表示返回。同时，使用了projection参数，则里面的要么全部为0，表示除了指明的键，其他全返回；要么全部为1，表示只返回指明的键。（_id默认返回，不返回需要标0）

```
> db.testcollection.find({}, {age : 1})
{ "_id" : ObjectId("617fa8afc8d545dc9dcfc9db"), "age" : 21 }
{ "_id" : ObjectId("617fa8c2c8d545dc9dcfc9dc") }

> db.testcollection.find({}, {name : 0})
{ "_id" : ObjectId("617fa8afc8d545dc9dcfc9db"), "age" : 21 }
{ "_id" : ObjectId("617fa8c2c8d545dc9dcfc9dc") }

> db.testcollection.find({}, {name : 0, _id : 0})
{ "age" : 21 }
{  }
```

- 特殊操作符**$type**根据类型返回特定值

```
# 常见Double:1, String:2, Boolean:8, Object:3, Array:4等其他可看官方文档
> db.testcollection.find({age:{$type:2}})
{ "_id" : ObjectId("617fa8c2c8d545dc9dcfc9dc"),"name":"JIA123", "age" : "ttt" }
```

- **limit()**指明需要返回多少条数据，**skip()**表示跳过前面多少条数据，两者结合可类似分页，但注意skip是一条一条数的，巨大数据量时效率低，可考虑根据其他字段的值进行比较分页。skip和limit的顺序无关，都是先进行skip的。

```
> db.testcollection.find({}).limit(1)
{ "_id" : ObjectId("617fa8afc8d545dc9dcfc9db"), "name" : "JIA", "age" : 21 }

> db.testcollection.find({}).limit(1).skip(1)
{ "_id" : ObjectId("617fa8c2c8d545dc9dcfc9dc"), "name":"JIA123", "age" : "ttt" }
```

- **sort()**排序，1/-1表示升序/降序。**sort()、limit()、skip()**三者联合使用时，无论顺序如何，先进行**sort()**，再进行**skip()**，最后进行**limit()**。

```
> db.testcollection.find({}).sort({name:-1})
{ "_id":ObjectId("617fa8c2c8d545dc9dcfc9dc"), "name" : "JIA123", "age" : "ttt" }
{ "_id":ObjectId("617fa8afc8d545dc9dcfc9db"), "name" : "JIA", "age" : 21 }

> db.testcollection.find({}).sort({name:-1}).limit(1)
{ "_id":ObjectId("617fa8c2c8d545dc9dcfc9dc"), "name" : "JIA123", "age" : "ttt" }
```

3.4）索引的使用

- **createIndex({keys}, [options])**，keys为创建索引的键，1/-1表示按升序/降序建立索引，可根据多个键建立索引。options为可选参数，说明如下：

Parameter	Type	Description
background	Boolean	建索引过程会阻塞其它数据库操作，background可指定以后台方式创建索引， 默认值为**false**。
unique	Boolean	建立的索引是否唯一。默认值为**false**.
name	string	索引的名称。如果未指定，MongoDB的通过连接索引的字段名和排序顺序生成一个索引名称。
sparse	Boolean	对文档中不存在的字段数据不启用索引；**这个参数需要特别注意，如果设置为true的话，在索引字段中不会查询出不包含对应字段的文档.。默认值为 false**.
expireAfterSeconds	integer	指定一个以秒为单位的数值，完成 TTL设定，设定集合的生存时间。
v	index version	索引的版本号。默认的索引版本取决于mongod创建索引时运行的版本。
weights	document	索引权重值，数值在 1 到 99,999 之间，表示该索引相对于其他索引字段的得分权重。
default_language	string	对于文本索引，该参数决定了停用词及词干和词器的规则的列表。 默认为英语
language_override	string	对于文本索引，该参数指定了包含在文档中的字段名，语言覆盖默认的language，默认值为 language

```
> db.testcollection.createIndex({name:1, age:-1}, {background:true})
{
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "createdCollectionAutomatically" : false,
        "ok" : 1
}
```

- 索引的查询与删除。**getIndexes()**查询所有索引，**totalIndexSize()**查询集合索引大小，**dropIndex(名称)和dropIndexes()**删除指定索引和全部索引

```
> db.testcollection.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_"
        },
        {
                "v" : 2,
```

```
                "key" : {
                        "name" : 1,
                        "age" : -1
                },
                "name" : "name_1_age_-1",
                "background" : true
        }
]

> db.testcollection.totalIndexSize()
45056

> db.testcollection.dropIndex("name_1_age_-1")
{ "nIndexesWas" : 2, "ok" : 1 }

# _id索引为基础索引，不会被删除
> db.testcollection.dropIndexes()
{
        "nIndexesWas" : 1,
        "msg" : "non-_id indexes dropped for collection",
        "ok" : 1
}
```

- 利用**expireAfterSeconds**选项对集合中的数据进行到时清除。仅限用于日期类字段，明面上是按照指定事件删除，但是文档是间隔事件扫描是否过期的，即删除不一定准时。

```
# 从clearDate字段的值表示的时间开始算起，180s后清除该文档。若为0则时间到了立即删除
> db.testcollection.createIndex({clearDate:1},{expireAfterSeconds:180})
{
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 3,
        "createdCollectionAutomatically" : false,
        "ok" : 1
}
```

3.5）聚合的使用

- MongoDB中聚合aggregate主要用于处理数据（如统计平均值，求和等），并返回计算后的结果。

3.5.1）管道

- 管道的概念类似于Java的Stream操作，即在管道中对文档进行处理，输出给下一个管道。需要注意的是，管道是有顺序的，在这里**skip、limit、sort**是按照顺序生效的。常用的几个管道如下：
  - $project：修改输入文档的结构。可以用来重命名、增加或删除域，也可以用于创建计算结果以及嵌套文档。用0表示不输出，1表示输出来选择字段。

    ```
    # 这里相当于只使用了集合中的age字段，默认_id是输出的，需要0来隐藏
    > db.testcollection.aggregate({$project:{age:1}})
    { "_id" : ObjectId("617fff5d1e8e67933849e40e"), "age" : 23 }
    { "_id" : ObjectId("617fff5d1e8e67933849e40f"), "age" : 24 }
    ```

  - $match：用于输出符合条件的文档

```
> db.testcollection.aggregate({$match:{age:{$gte:23}}})
{ "_id" : ObjectId("617fff5d1e8e67933849e40e"), "name" : "JIA", "age" : 23 }
{ "_id" : ObjectId("617fff5d1e8e67933849e40f"), "name" : "JIA123", "age" :
24 }
```

- $skip、$limit、$sort与查询时效果一致

```
> db.testcollection.aggregate({$skip: 1})
{ "_id" : ObjectId("617fff5d1e8e67933849e40f"), "name" : "JIA123", "age" :
24 }

> db.testcollection.aggregate({$sort: {age: -1}})
{ "_id" : ObjectId("617fff5d1e8e67933849e40f"), "name" : "JIA123", "age" :
24 }
{ "_id" : ObjectId("617fff5d1e8e67933849e40e"), "name" : "JIA", "age" : 23 }

> db.testcollection.aggregate({$limit: 1})
{ "_id" : ObjectId("617fff5d1e8e67933849e40e"), "name" : "JIA", "age" : 23 }
```

- $group：用于分组，并可以进行计算，使用格式$group : {_id: "", 输出字段名 : {操作}}。其中_id字段必须存在，是用于分组的标志，如按age分组则使用_id : "$age"，若不分组则可以_id : null。

```
> db.testcollection.aggregate({$group:{_id: "$name", age_sum: {$sum: "$age"
}}})
{ "_id" : "JIA123", "age_sum" : 24 }
{ "_id" : "JIA", "age_sum" : 48 }
```

- $unwind：将数组字段拆开成为一系列文档

```
> db.testcollection.insert({name:"ZURI",ip:[192,168,0,1]})
WriteResult({ "nInserted" : 1 })
> db.testcollection.aggregate([{$match:{name:'ZURI'}}, {$unwind:'$ip'}])
{ "_id" : ObjectId("6180142b1e8e67933849e411"), "name" : "ZURI", "ip" : 192
}
{ "_id" : ObjectId("6180142b1e8e67933849e411"), "name" : "ZURI", "ip" : 168
}
{ "_id" : ObjectId("6180142b1e8e67933849e411"), "name" : "ZURI", "ip" : 0 }
{ "_id" : ObjectId("6180142b1e8e67933849e411"), "name" : "ZURI", "ip" : 1 }
```

3.5.2）group内的操作

表达式	描述	实例
$sum	计算总和。	$group : {_id : "$name", age_sum : {$sum : "$age"}}
$avg	计算平均值	$group : {_id : "$name", age_avg : {$avg : "$age"}}
$min	获取集合中所有文档对应值得最小值。	$group : {_id : "$name", age_min : {$min : "$age"}}
$max	获取集合中所有文档对应值得最大值。	$group : {_id : "$name", age_max : {$max : "$age"}}
$push	将值加入一个数组中，不判断是否有重复的值。	$group : {_id : "$name", age_array : {$push: "$age"}}
$addToSet	将值加入一个数组中，会判断是否有重复的值，若相同的值在数组中已经存在了，则不加入。	$group : {_id : "$name", age_array : {$addToSet : "$age"}}
$first	根据资源文档的排序获取第一个文档数据。	$group : {_id : "$name", first_age : {$first : "$age"}}
$last	根据资源文档的排序获取最后一个文档数据	$group : {_id : "$name", last_age : {$last : "$age"}}

3.6）MongoDB的复制/分片

- 主节点记录在其上的所有操作oplog，从节点定期轮询主节点获取这些操作，然后对自己的数据副本执行这些操作，从而保证从节点的数据与主节点一致。
- 副本集特征：任何节点都可以成为主节点，所有写入操作都在主节点上，自动故障转移和自动恢复。
- 分片：当MongoDB存储海量的数据时，一台机器可能不足以存储数据，也可能不足以提供可接受的读写吞吐量。这时，我们就可以通过在多台机器上分割数据，使得数据库系统能存储和处理更多的数据。
- 搭建过程参考网上资料

3.7）MongoDB的备份与恢复

- **mongodump**与**mongorestore**用于备份与恢复，其中**mongodump -h dbhost -d dbname -o dbdirectory**命令用于备份指定数据库到指定目录下，而**mongorestore -h dbhost -d dbname path/-dir**用于恢复数据库。

3.8）MongoDB监控

- bin下提供了**mongostat**与**mongotop**命令用于查看当前的MongoDB服务的状态，其中**mongostat**会间隔固定时间获取mongodb的当前运行状态，**mongotop**提供每个集合的水平的统计数据，跟踪MongoDB的实例，查看哪些大量的时间花费在读取和写入数据

3.9）MongoDB结合JAVA、Spring Boot

- 结合Java，使用依赖包`mongo-driver-java`，使用`MongoClient`建立连接，`MongoDatabase`指定数据库，并在`MongoDatabase`对象上使用MongoDB指令即可。
- 结合Spring Boot，使用依赖包`spring-boot-starter-data-mongodb`，在配置文件中配置MongoDB链接地址`spring.data.mongodb.uri`，Spring Data Mongo提供了一个MongoTemplate，用于操作，实体类映射是通过`MongoMappingConverter`这个类实现的。

4）MongoDB的高级使用

4.1）MongoDB的关系

- 嵌入型关系：直接将关联的数据存在一个文档中，如把address文档存在user文档中，查询时可以直接根据user文档检索其地址。

```
>db.usercollection.findOne({"name":"JIA"},{"address":1})
```

- 关联型关系：将关联的文档的_id字段存放在文档中，需要时再去另一个集合检索。相当于外键。

```
>var result = db.usercollection.findOne({"name":"JIA"},{"address_ids":1})
>var addresses = db.addresscollcetion.find({"_id":
{"$in":result["address_ids"]}})
```

4.2）MongoDB的引用

- 除了上面这种简单的使用_id作为关联关系之外，MongoDB还提供了**DBRefs**更方便的指明引用。其形式为**{ $ref, $id, $db}**，其中$ref表示所引用的集合名称，$id表示所引用的id，$db表示该集合所在的数据库（可选参数）。

```
>db.testcollection.insert({name: "JIA", address: {$ref:"address_home",
$id:ObjectId("6180c6f45eef60d2b4c6d29c")}})

# 生成的文档如下
{
        "_id" : ObjectId("6180c8425eef60d2b4c6d29e"),
        "name" : "JIA",
        "address" : DBRef("address_home", ObjectId("6180c6f45eef60d2b4c6d29c"))
}

> db[addressRef.$ref].findOne({_id:addressRef.$id})
{ "_id" : ObjectId("6180c6f45eef60d2b4c6d29c"), "address" : "China" }
```

4.3）MongoDB的索引覆盖

- 当出现以下情况时，查询会使用到索引。因为索引存在于RAM中，从索引中获取数据比通过扫描文档读取数据要快得多。能覆盖查询的是以下的查询：
    - 所有的查询字段是索引的一部分，如建立了name和gender字段的索引，当查询返回的字段是name或者gender（亦或这两个都有时），可以直接从索引中取得数据。（查询默认返回_id，需要指定_id:0不返回，不然无法覆盖索引）

    - 所有的查询返回字段在同一个索引中，如建立了name、gender、address索引，当查询返回的字段是上诉的情况时，也会覆盖索引。
- 但是，当索引字段在集合中只是数组中的一部分时，不会覆盖索引。

4.4）MongoDB的查询分析

- **explain()**可以用于分析是否使用了索引等信息，**hint()**可以强制使用指定的索引

```
>db.usercollection.find({gender:"M"},{name:1,_id:0}).explain()
>db.usercollection.find({gender:"M"},
{name:1,_id:0}).hint({gender:1,name:1}).explain()
```

4.5）MongoDB的原子操作

- 虽然MongoDB**不支持事务**，但它还是提供了部分原子操作的。除了下面的findAndModify()，还有许多如$set、$pushAll等。

```
# 查询符合条件的再进行修改
db.books.findAndModify ( {
   query: {
            _id: ObjectId("123456789"),
            available: { $gt: 0 }
          },
   update: {
             $inc: { available: -1 },
             $push: { checkout: { by: "abc", date: new Date() } }
           }
} )
```

4.6）MongoDB的高级索引

- 索引数组字段：在数组中创建索引，需要对数组中的每个字段依次建立索引。所以在我们为数组tags 创建索引时，会为 music、cricket、blogs三个值建立单独的索引。（假定文档存在数组tags : ["music", "cricket", "blogs"])

```
>db.usercollection.createIndex({"tags":1})
>db.usercollection.find({tags:"cricket"})
```

- 索引子文档字段：对子文档中的字段进行创建索引时，需要注明来自哪个子文档的字段。（假定文档存在子文档address，其三个字段为city、state、pincode）

```
>db.usercollection.createIndex({"address.city":1,"address.state":1,"address.pinc
ode":1})
>db.usercollection.find({"address.city":"Los Angeles"})
# 查询表达不一定遵循指定的索引的顺序，mongodb 会自动优化
>db.usercollection.find({"address.state":"California","address.city":"Los
Angeles"})
```

4.7）MongoDB的索引限制

- 索引不能被以下的查询使用：
  - 正则表达式及非操作符，如 $nin, $not, 等。
  - 算术运算符，如 $mod, 等。
  - $where 子句
- 内存限制：由于索引是存储在内存(RAM)中，应该确保该索引的大小不超过内存的限制。
- 最大范围：
  - 集合中索引不能超过64个
  - 索引名的长度不能超过128个字符
  - 一个复合索引最多可以有31个字段
- 如果文档的索引字段值超过了索引键的限制，MongoDB不会将任何文档转换成索引的集合。

4.8）MongoDB的Map-Reduce

- Map-Reduce是一种计算模型，简单的说就是将大批量的工作（数据）分解（MAP）执行，然后再将结果合并成最终结果（REDUCE）。基本格式如下：

```
>db.collection.mapReduce(
   function() {emit(key,value);},  //map 函数
   function(key,values) {return reduceFunction},   //reduce 函数
   {
      out: collection,
      query: document,
      sort: document,
      limit: number
   }
)
```

- 其中，Map函数调用 emit(key, value)，遍历 collection 中所有的记录，返回键值对。其他参数说明如下：
  - **reduce**： 统计函数，reduce函数的任务就是将key-values变成key-value，也就是把values数组变成一个单一的值value。
  - **out**：统计结果存放集合 (不指定则使用临时集合（inline：1），在客户端断开后自动删除)。
  - **query**： 一个筛选条件，只有满足条件的文档才会调用map函数。（query，limit，sort可以随意组合）
  - **sort**：和limit结合的sort排序参数（也是在发往map函数前给文档排序），可以优化分组机制
  - **limit**：发往map函数的文档数量的上限（要是没有limit，单独使用sort的用处不大）

```
> db.testcollection.mapReduce(
    function() {emit(this.age, this.name);},
    function(key, values) {
        let result = "";
        for (let index in values){
            result = result + " " + values[index];
        }
        return result;
    },
    {
        query: {status:true},
        out: {inline:1}
    }
```

```
)
# 结果
{
        "results" : [
                {
                        "_id" : 20,
                        "value" : " zuri jia ZURI JIA"
                }
        ],
        "ok" : 1
}
```

4.9）MongoDB的全文检索

- 全文检索对每一个词建立一个索引，指明该词在文章中出现的次数和位置，当用户查询时，检索程序根据事先建立的索引进行查找，并将查找的结果反馈给用户。
- 创建全文索引，普通索引使用1/-1，全文索引使用“text”。查询时使用$text和$search

```
> db.article.createIndex({content:"text"})
> db.article.find({$text:{$search:"hello"}})
```

4.10）MongoDB的正则表达式使用

- **$regex**如`db.posts.find({post_text:{$regex:"runoob",$options:"$i"}})`这种情况下将会查询包含runoob的数据，且options里设定了忽略大小写。