MongoDB介绍

MongoDB简介

MongoDB 是免费开源的跨平台 NoSQL 数据库，命名源于英文单词 hu**mongo**us，意思是「巨大无比」，可见开发组对 MongoDB 的定位。与关系型数据库不同，MongoDB 的数据以类似于 JSON 格式的二进制文档存储：

```
{
    name: "McConaughey",
    age: 18,
    hobbies: ["travel", "swimming"]
}
```

文档型的数据存储方式有几个重要好处：文档的数据类型可以对应到语言的数据类型，如数组类型（Array）和对象类型（Object）；文档可以嵌套，有时关系型数据库涉及几个表的操作，在MongoDB 中一次就能完成，可以减少昂贵的连接花销；文档不对数据结构加以限制，不同的数据结构可以存储在同一张表。

MongoDB的适用场景

网站数据：Mongo 非常适合实时的插入,更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。

缓存：由于性能很高，Mongo 也适合作为信息基础设施的缓存层。在系统重启之后，由Mongo搭建的持久化缓存层可以避免下层的数据源过载。

大尺寸、低价值的数据：使用传统的关系型数据库存储一些大尺寸低价值数据时会比较浪费，在此之前，很多时候程序员往往会选择传统的文件进行存储。

高伸缩性的场景：Mongo 非常适合由数十或数百台服务器组成的数据库，Mongo 的路线图中已经包含对MapReduce 引擎的内置支持以及集群高可用的解决方案。

用于对象及JSON 数据的存储：Mongo 的BSON 数据格式非常适合文档化格式的存储及查询。

- MongoDB的行业具体应用

游戏场景，使用 MongoDB 存储游戏用户信息，用户的装备、积分等直接以内嵌文档的形式存储，方便查询、更新。

物流场景，使用 MongoDB 存储订单信息，订单状态在运送过程中会不断更新，以 MongoDB 内嵌数组的形式来存储，一次查询就能将订单所有的变更读取出来。

社交场景，使用 MongoDB 存储存储用户信息，以及用户发表的朋友圈信息，通过地理位置索引实现附近的人、地点等功能。

物联网场景，使用 MongoDB 存储所有接入的智能设备信息，以及设备汇报的日志信息，并对这些信息进行多维度的分析。

直播，使用 MongoDB 存储用户信息、礼物信息等。

- 如何抉择是否使用MongoDB

应用特征	Yes/No
应用不需要复杂事务及复杂join支持	必须yes
新应用，需求会变，数据模型无法确定，想快速迭代开发	?
应用需要2000-3000以上的读写QPS（更高也可以）	?
应用需要TB甚至 PB 级别数据存储	?
应用发展迅速，需要能快速水平扩展	?
应用需要大量的地理位置查询、文本查询	?
应用需要99.999%高可用	?

有一个yes就可以选择MongoDB，两个以上yes，选MongoDB绝对不后悔

实时效果反馈

1.MongoDB采用什么形式存储数据?

A 二维表格

B key-value

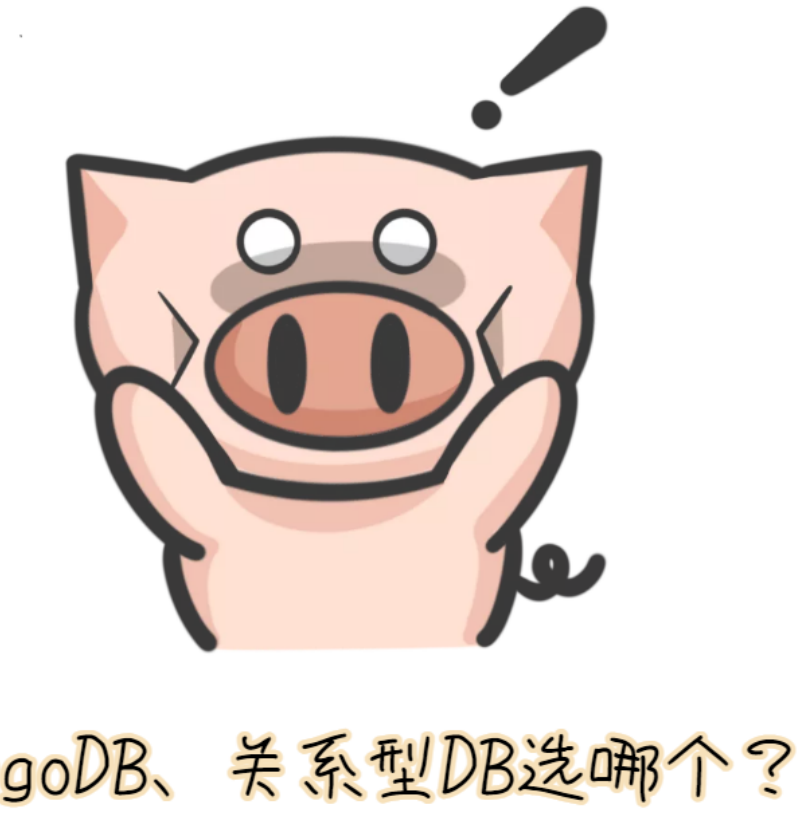
C 文档

D 以上都是

答案

1=>C

与关系型数据库比较



MongoDB与RDMS(关系型数据库)比较，如下图所示

RDMS	MongoDB
database（数据库）	database（数据库）
table （表）	collection（ 集合）
row（ 行）	document（ BSON 文档）
column （列）	field （字段）
index（唯一索引、主键索引）	index （支持地理位置索引、全文索引 、哈希索引）
join （主外键关联）	embedded Document (嵌套文档)
primary key(指定1至N个列做主键)	primary key (指定_id field做为主键)

什么是BSON

BSON是一种类似于JSON的二进制形式的存储格式，简称Binary JSON，它和JSON一样，支持内嵌的文档对象和数组对象，但是BSON有JSON没有的一些数据类型，如Date和BinData类型。BSON有三个特点：轻量性、可遍历性、高效性。

下表列出了MongoDB中Document可以出现的数据类型：

数据类型	说明	document举例
String	字符串	{key:“cba”}
Integer	整型数值	{key:2}
Boolean	布尔型	{key:true}
Double	双精度浮点数	{key:0.23}
ObjectId	对象id，用于创建文档的id	{_id:new ObjectId()}
Array	数组	{arr:[“jack”,“tom”]}
Timestamp	时间戳	{ createTime: new Timestamp() }
object	内嵌文档	{student:{name:“zhangsan”,age:18}}
null	空值	{key:null}
Date或者ISODate	日期时间	{birthday:new Date()}
Code	代码	{setPersonInfo:function(){}}

实时效果反馈

1.MongoDB的特点不包括哪个?

A 读取速度快

B 支持地理坐标索引

C 以BSON文档为单位进行存储

D 支持事务

答案

1=>D

MongoDB安装

- 独立安装

进入/etc/yum.repos.d目录，创建文件mongodb-org-5.0.repo

```
[root@192 yum.repos.d]# pwd
/etc/yum.repos.d
[root@192 yum.repos.d]#
```

在文件内输入yum源地址

```
[mongodb-org]
name=MongoDB Repository
baseurl=http://mirrors.aliyun.com/mongodb/yum/redhat/7Server/mongodb-org/5.0/x86_64/
gpgcheck=0
enabled=1
```

安装之前先更新

```
yum update
```

开始安装

```
yum -y install mongodb-org
```

安装完成后，查看mongo的安装位置

```
whereis mongod
```

修改配置文件

```
vim /etc/mongod.conf
```

```
bindIp: 172.0.0.1 改为 bindIp: 0.0.0.0
```

启动

```
systemctl start mongod.service
```

停止

```
systemctl stop mongod.service
```

重启

```
systemctl restart mongod.service
```

查看

```
systemctl status mongod.service
```

设置开机自启动

```
systemctl enable mongod.service
```

mongo shell 的启动

```
./bin/mongo
```

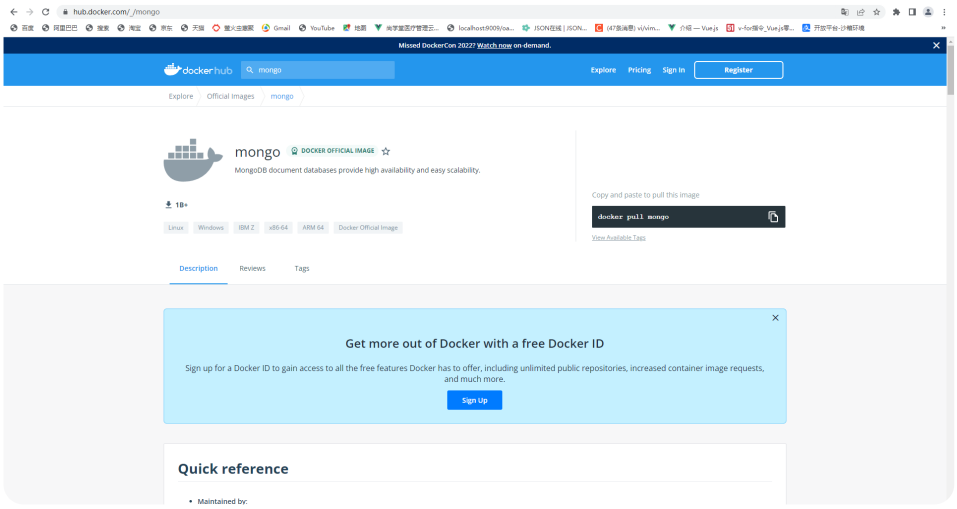
指定主机和端口的方式启动

```
./bin/mongo --host=主机IP --port=端口
```

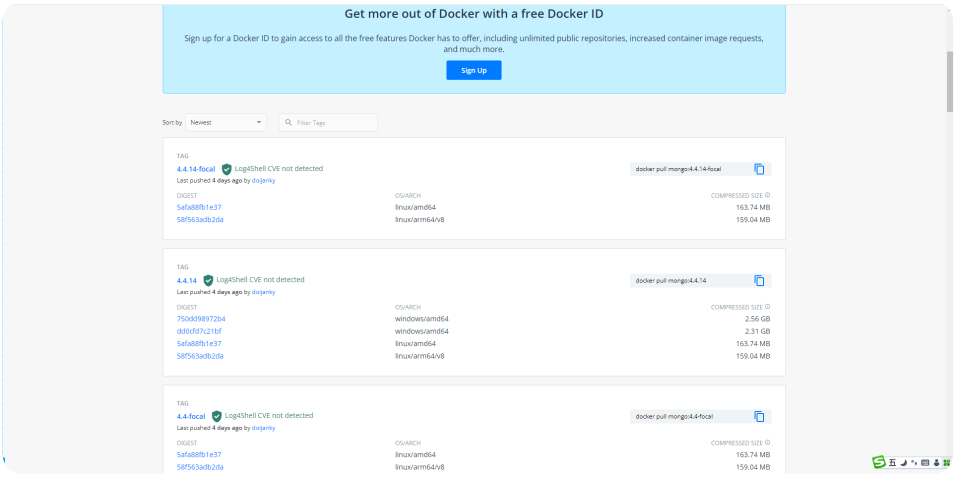
```
Enable MongoDB's free cloud-based monitoring service, which will then receive and displa
y
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible
to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitorin
g()
---

>
```

- docker安装

访问hub.docker.com，搜索mongo镜像



查看可用的镜像



获取你想要拉取的镜像

```
docker pull mongo:xxx
```

查看已下载的镜像

```
docker images
```

创建挂载目录

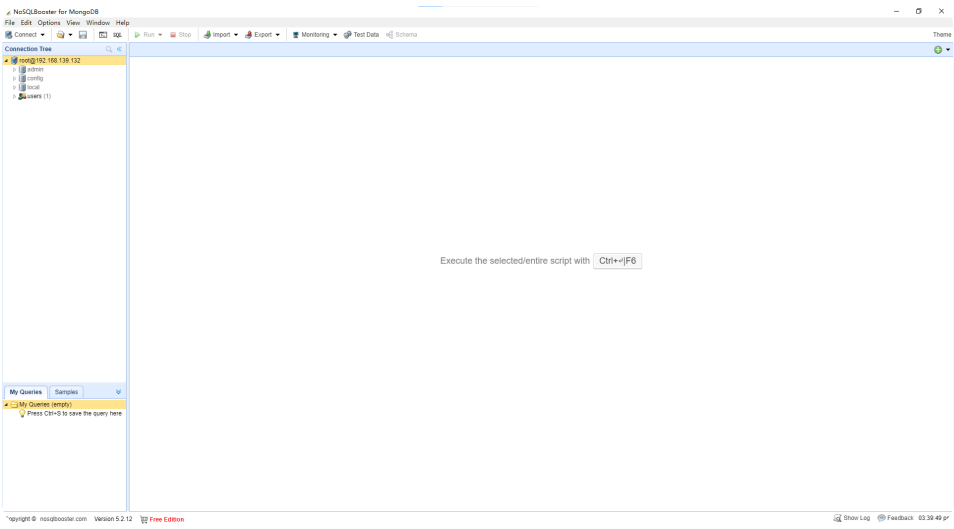
```
mkdir -p /mnt/mongodb/data  --创建保存数据目录
```

创建并运行 mongo 容器

```
docker run -itd --privileged=true --name mongo5
-p 27017:27017 -v /mnt/mongodb/data:/data/db
mongo:5.0.9-focal
```

- 使用GUI工具连接MongoDB

使用NosqlBooster For MongoDB客户端工具连接MongoDB数据库



实时效果反馈

1.MongoDB默认端口是?

A 6379

B 27017

C 8761

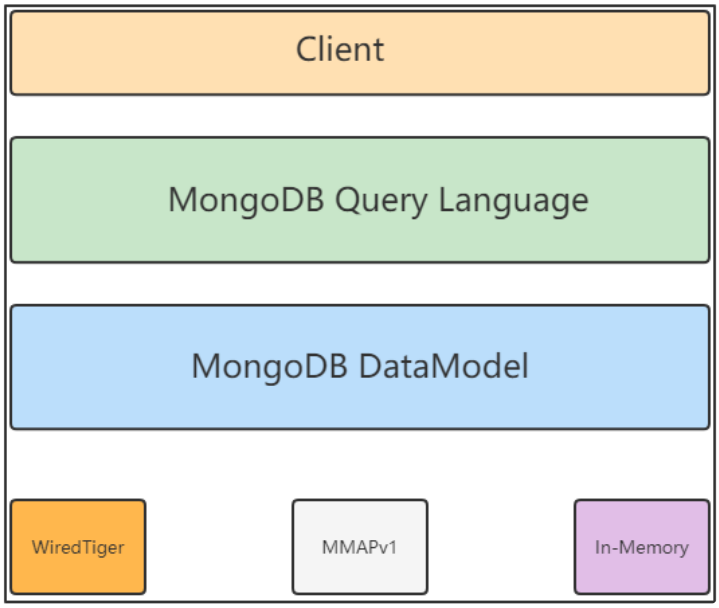
D 15672

答案

1=>B

MongoDB架构

MongoDB逻辑结构



MongoDB 与 MySQL 中的架构相差不多，底层都使用了可插拔的存储引擎以满足用户的不同需要。用户可以根据程序的数据特征选择不同的存储引擎，在最新版本的 MongoDB 中使用了 WiredTiger 作为默认的存储引擎，WiredTiger 提供了不同粒度的并发控制和压

缩机制，能够为不同种类的应用提供了最好的性能和存储率。

在存储引擎上层的就是 MongoDB 的数据模型和查询语言了，由于 MongoDB 对数据的存储与 RDBMS有较大的差异，所以它创建了一套不同的数据模型和查询语言。

MongoDB数据模型

与SQL数据库不同，在SQL数据库中，在插入数据之前必须确定和声明表的架构，默认情况下，MongoDB的集合不要求其文档具有相同的架构。

- 结构灵活

单个集合中的文档不需要具有相同的字段集合，并且集合中的文档之间的字段数据类型可能不同；

可灵活更改集合中文档的结构，如添加新字段、删除现有字段或将字段值更改为新类型，将文档更新为新结构。

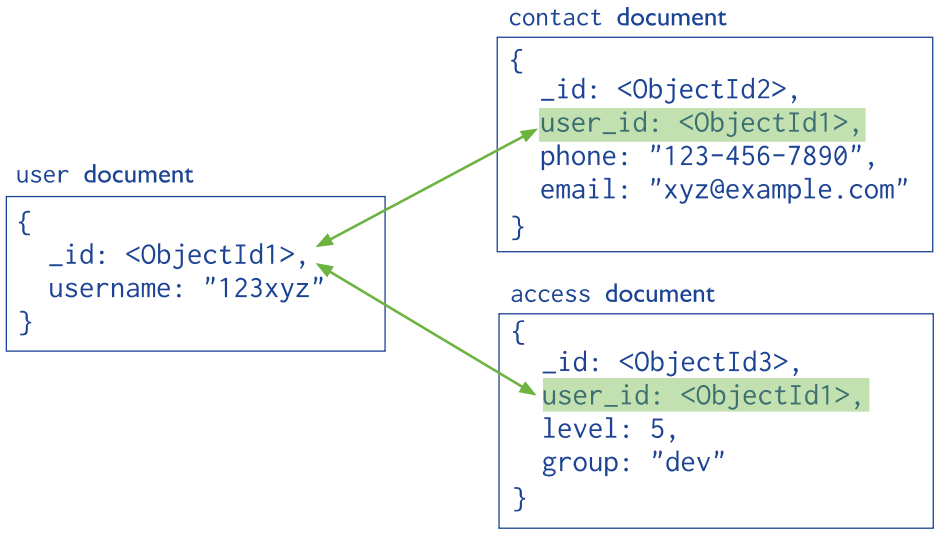
- 文档结构

内嵌

内嵌的方式指的是把相关联的数据保存在同一个文档结构之中。MongoDB的文档结构允许一个字段或者一个数组内的值作为一个嵌套的文档。

引用

引用方式通过存储数据引用信息来实现两个不同文档之间的关联,应用程序可以通过解析这些数据引用来访问相关数据。



如何选择数据模型

- 选择内嵌:

数据对象之间有包含关系 ,一般是数据对象之间有一对多或者一对一的关系 。

需要经常一起读取的数据。

有 map-reduce/aggregation 需求的数据放在一起，这些操作都只能操作单个 collection。

- 选择引用:

当内嵌数据会导致很多数据的重复，并且读性能的优势又不足于覆盖数据重复的弊端 。

需要表达比较复杂的多对多关系的时候 。

大型层次结果数据集 嵌套不要太深。

实时效果反馈

1.MongoDB集合中一个集合下所有文档的结构必须一致?

A 必须一致

B 必须不一致

C 可以不一致

答案

1=>C

MongoDB存储引擎

存储引擎负责管理数据如何在磁盘和内存中存储，MongoDB支持多种存储引擎，不同的存储引擎都有其在特定工作环境下的特点，选择适合的存储引擎能够有效提高应用程序的性能。

WiredTiger存储引擎 (默认)

WiredTiger适合大多数工作场景，从MongoDB 3.2开始默认的存储引擎是WiredTiger，3.2版本之前的默认存储引擎是MMAPv1，MongoDB 4.x版本不再支持MMAPv1存储引擎。

- 支持document级别并发操作

WiredTiger对写入操作使用document级并发控制。因此，多个客户端可以同时修改集合的不同文档。对于大多数读写操作，WiredTiger使用乐观并发控制。WiredTiger仅在全局、数据库和集合级别使用意向锁。当存储引擎检测到两个操作之间的冲突时，其中一个操作将引发写入冲突，MongoDB会对用户透明地重试该操作。

- Snapshots and Checkpoints（快照和检查点）

WiredTiger使用多版本并发控制（MVCC）。在操作开始时，WiredTiger会向操作提供数据的point-in-time快照，快照显示了数据在内存中的一致性视图。

从3.6版开始，MongoDB将WiredTiger配置为每隔60秒创建检查点（即将快照数据写入磁盘）。在早期版本中，MongoDB将检查点设置为在WiredTiger中每隔60秒或写入2 GB日志数据时（以先发生的为准）对用户数据进行检查。

- Journal（日志）

WiredTiger将日志与检查点结合使用，以确保数据的持久性。WiredTiger日志将保留检查点之间的所有数据修改。如果MongoDB在两个检查点之间退出，它将使用日志重播自上一个检查点以来修改的所有数据

- Compression（压缩）

使用WiredTiger存储引擎，MongoDB会对所有集合和索引进行压缩，压缩以牺牲额外的CPU为代价，最大限度地减少了磁盘的使用。默认情况下，WiredTiger对所有集合使用block compression，并对索引使用prefix compression。

- 内存使用

对于WiredTiger，MongoDB利用WiredTiger内部缓存和文件系统缓存。从MongoDB 3.4开始，默认WiredTiger内部缓存大小为以下两者中的较大值：

50% of (RAM - 1 GB)

256 MB

实时效果反馈

1.MongoDB4.4使用的存储引擎是?

A MyISAM

B InnoDB

C WiredTiger

D MMAPv1

答案

1=>C

2.一台主机的内存是1.25GB时，wiredTiger存储引擎的内部缓存大小是?

A 0.5GB

B 256MB

C 1GB

D 以上都不对

答案

2=>B

In-memory存储引擎

从MongoDB 企业版3.2.6版开始，In-Memory存储引擎是64位版本中广泛使用（general availability GA）的一部分。除某些元数据和诊断数据外，In-Memory存储引擎不维护任何磁盘上的数据，包括配置数据，索引，用户凭据等。

- In-Memory存储引擎设置

配置--storageEngine选项值为inMemory；如果使用配置文件，则配置storage.engine

配置--dbpath，如果使用配置文件则配置storage.dbPath。尽管In-Memory存储引擎不会将数据写入文件系统，但它会在--dbpath中维护小型元数据文件和诊断数据以及用于构建大型索引的临时文件。

```
mongod  --storageEngine inMemory  --dbpath
<path>
```

```
storage:
   engine: inMemory
   dbPath: <path>
```

- 并发(concurrency)

In-Memory存储引擎对于写入操作使用了document级并发控制。多个客户端可以同时修改集合的不同文档。

- 内存使用

默认情况下，In-Memory存储引擎使用50%的物理RAM减去1 GB。如果写操作的数据超过了指定的内存大小，则MongoDB返回错误：

```
"WT_CACHE_FULL: operation would overflow cache"
```

要指定新大小，可使用YAML格式配置文件的

```
storage:
   engine: inMemory
   dbPath: <path>
   inMemory:
      engineConfig:
         inMemorySizeGB: <newSize>
```

- 事务

从MongoDB 4.2开始，复制集和分片集群上支持事务，其中：主成员节点使用WiredTiger存储引擎，同时，辅助成员使用WiredTiger存储引擎或In-Memory存储引擎。
在MongoDB 4.0中，只有使用WiredTiger存储引擎的复制集才支持事务。

实时效果反馈

1.In-memory存储引擎不持久化的数据有?

A 应用数据

B 索引

C 用户凭据

D 以上都是

答案

1=>D

MongoDB命令

基本操作

查看数据库

```
show dbs;
```

切换数据库 如果没有对应的数据库则创建

```
use 数据库名;
```

创建集合

```
db.createCollection("集合名")
```

查看集合

```
show tables;
show collections;
```

删除集合

```
db.集合名.drop();
```

删除当前数据库

```
db.dropDatabase();
```

实时效果反馈

1.MongoDB查看集合的命令是?

A select * from collection

B use collection

C show collections

D 以上都不对

答案

1=>C

CRUD操作

添加文档

添加单个文档，如果集合不存在，会创建一个集合

```
db.collection.insertOne()
```

如果不指定id， MongoDB会使用ObjectId的value作为id

```
db.inventory.insertOne(
{ item: "canvas", qty: 100, tags: ["cotton"],
size: { h: 28, w: 35.5, uom: "cm" } })
```

添加多个文档

```
db.collection.insertMany()
```

```
db.inventory.insertMany([
   { item: "journal", qty: 25, tags: ["blank",
"red"], size: { h: 14, w: 21, uom: "cm" } },
   { item: "mat", qty: 85, tags: ["gray"],
size: { h: 27.9, w: 35.5, uom: "cm" } },
   { item: "mousepad", qty: 25, tags: ["gel",
"blue"], size: { h: 19, w: 22.85, uom: "cm"
}}])
```

查询文档

首先插入一批文档，再进行查询

```
db.inventory.insertMany( [
   { item: "journal", qty: 25, size: { h: 14,
w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5,
w: 11, uom: "in" }, status: "A" },
   { item: "paper", qty: 100, size: { h: 8.5,
w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h:
22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10,
w: 15.25, uom: "cm" }, status: "A" }
]);
```

查询集合所有文档

```
db.inventory.find({})
```

查询指定内容的文档，匹配1条

```
db.inventory.find( { size: { h: 14, w: 21, uom: "cm" } } )
```

匹配0条，原因是顺序不匹配

```
db.inventory.find(  { size: { w: 21, h: 14, uom: "cm" } }  )
```

匹配size中uom属性为“in”的文档

```
db.inventory.find( { "size.uom": "in" } )
```

匹配size中h属性值小于15的文档

```
db.inventory.find( { "size.h": { $lt: 15 } } )
```

匹配h属性小于15并且uom属性为“in”，并且“status”属性为"D"的文档

```
db.inventory.find( { "size.h": { $lt: 15 }, "size.uom": "in", status: "D" } )
```

更新文档

插入下列文档，供更新操作

```
db.inventory.insertMany( [
   { item: "canvas", qty: 100, size: { h: 28,
w: 35.5, uom: "cm" }, status: "A" },{ item:
"journal", qty: 25, size: { h: 14, w: 21, uom:
"cm" }, status: "A" },{ item: "mat", qty: 85,
size: { h: 27.9, w: 35.5, uom: "cm" }, status:
"A" },{ item: "mousepad", qty: 25, size: { h:
19, w: 22.85, uom: "cm" }, status: "P" },
{ item: "notebook", qty: 50, size: { h: 8.5, w:
11, uom: "in" }, status: "P" },{ item: "paper",
qty: 100, size: { h: 8.5, w: 11, uom: "in" },
status: "D" },{ item: "planner", qty: 75, size:
{ h: 22.85, w: 30, uom: "cm" }, status: "D" },{
item: "postcard", qty: 45, size: { h: 10, w:
15.25, uom: "cm" }, status: "A" },
{ item: "sketchbook", qty: 80, size: { h: 14,
w: 21, uom: "cm" }, status: "A" },{ item:
"sketch pad", qty: 95, size: { h: 22.85, w:
30.5, uom: "cm" }, status: "A" }] );
```

更新item值为“paper”的第一个文档
将它的size.uom设置为“cm”,status值设置为"P"
并且把lastModified字段更新为当前时间，如果该字段不存在，则生成一个

```
db.inventory.updateOne(
   { item: "paper" },
   {$set: { "size.uom": "cm", status: "P" },
    $currentDate: { lastModified: true }}
)
```

更新qty属性值小于50的文档
将它的size.uom设置为“in”,status值设置为"P"
并且把lastModified字段更新为当前时间，如果该字段不存在，则生成一个

```
db.inventory.updateMany(
   { "qty": { $lt: 50 } },
   { $set: { "size.uom": "in", status: "P" },
     $currentDate: { lastModified: true }
   }
)
```

把item属性为“paper”的文档替换成下面的内容

```
db.inventory.replaceOne(
   { item: "paper" },{ item: "paper", instock:
[ { warehouse: "A", qty: 60 }, { warehouse:
"B", qty: 40 } ] })
```

删除文档

删除集合所有文档

```
db.inventory.deleteMany({})
```

删除指定条件的文档

```
db.inventory.deleteMany({ status : "A" })
```

最多删除1个指定条件的文档

```
db.inventory.deleteOne( { status: "D" } )
```

与SQL语法的对应

SQL Schema Statements	MongoDB Schema Statements

`CREATE TABLE people (`
`  id MEDIUMINT NOT NULL`
`  AUTO_INCREMENT,`
`  user_id Varchar(30),`
`  age Number,`
`  status char(1),`
`  PRIMARY KEY (id)`
`)`
	Implicitly created on first `insertOne()` or `insertMany()` operation. The primary key `_id` is automatically added if `_id` field is not specified.
	`db.people.insertOne( {`
	`  user_id: "abc123",`
	`  age: 55,`
	`  status: "A"`
	`} )`
	However, you can also explicitly create a collection:
	`db.createCollection("people")`

`ALTER TABLE people`
`ADD join_date DATETIME`
	Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.
	However, at the document level, `updateMany()` operations can add fields to existing documents using the `$set` operator.
	`db.people.updateMany(`
	`  { },`
	`  { $set: { join_date: new Date() } }`
	`)`

`ALTER TABLE people`
`DROP COLUMN join_date`
	Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.
	However, at the document level, `updateMany()` operations can remove fields from documents using the `$unset` operator.
	`db.people.updateMany(`
	`  { },`
	`  { $unset: { "join_date": "" } }`
	`)`

`CREATE INDEX idx_user_id_asc` `ON people(user_id)`	`db.people.createIndex( { user_id: 1 } )`

`CREATE INDEX`
`  idx_user_id_asc_age_desc`
`ON people(user_id, age DESC)`
	`db.people.createIndex( { user_id: 1, age:`

`DROP TABLE people`	`db.people.drop()`

实时效果反馈

1.MongoDB插入单个文档命令是?

A db.collection.insertMany()

B db.collection.insertOne()

C db.collection.find()

D 以上都不对

答案

1=>B

实时效果反馈

2.MongoDB查询集合所有文档的命令是?

A db.collection.find({id:0})

B db.collection.findAll()

C db.collection.find({})

D 以上都不对

答案

2=>C

实时效果反馈

3.MongoDB更新文档的命令不包括?

A db.collection.updateOne()

B db.collection.updateMany()

C db.collection.replaceOne()

D db.collection.replaceMany()

答案

3=>D

实时效果反馈

4.MongoDB删除集合所有文档的命令?

A db.collection.deleteOne()

B db.collection.deleteMany()

C db.collection.deleteAll()

D db.collection.deleteMany({})

答案

4=>D

聚合操作

通过聚合操作可以处理多个文档，并返回计算后的结果。

- 对多个文档进行分组

- 对分组的文档执行操作并返回单个结果
- 分析数据变化

聚合管道

分别由多个阶段来处理文档，每个阶段的输出是下个阶段的输入，返回的是一组文档的处理结果，例如，total、average、maxmium、minimium。

插入下列数据，供聚合操作用

```
db.orders.insertMany( [
   { _id: 0, name: "Pepperoni", size: "small",
price: 19,
     quantity: 10, date: ISODate( "2030-03-
13T08:14:30Z" ) },
   { _id: 1, name: "Pepperoni", size: "medium",
price: 20,
     quantity: 20, date : ISODate( "2030-03-
13T09:13:24Z" ) },
   { _id: 2, name: "Pepperoni", size: "large",
price: 21,
     quantity: 30, date : ISODate( "2030-03-
17T09:22:12Z" ) },
   { _id: 3, name: "Cheese", size: "small",
price: 12,
     quantity: 15, date : ISODate( "2030-03-
13T11:21:39.736Z" ) },{ _id: 4, name: "Cheese",
size: "medium", price: 13,
     quantity:50, date : ISODate( "2031-01-
12T21:23:13.331Z" ) }, { _id: 5, name:
"Cheese", size: "large", price: 14,
     quantity: 10, date : ISODate( "2031-01-
12T05:08:13Z" ) },
```

```
   { _id: 6, name: "Vegan", size: "small",
price: 17,
     quantity: 10, date : ISODate( "2030-01-
13T05:08:13Z" ) },
   { _id: 7, name: "Vegan", size: "medium",
price: 18,
     quantity: 10, date : ISODate( "2030-01-
13T05:10:13Z" ) }
] )
```

计算尺寸为medium的订单中，每种类型的订单数量

```
db.orders.aggregate( [
   // Stage 1: 匹配size:"medium"的文档
   {
      $match: { size: "medium" }
   },
   // Stage 2: 根据name统计过滤后的文档，并
把"quantity"值相加
   {
      $group: { _id: "$name", totalQuantity: {
$sum: "$quantity" } }
   }
] )
```

输出结果:

```
[
   { _id: 'Cheese', totalQuantity: 50 },
   { _id: 'Vegan', totalQuantity: 10 },
   { _id: 'Pepperoni', totalQuantity: 20 }
]
```

更复杂的例子：

```
db.orders.aggregate( [
   // Stage 1: 根据日期范围过滤
   {
      $match:
      {
         "date": { $gte: new ISODate( "2030-01-01" ), $lt: new ISODate( "2030-01-30" ) }
      }
   },
   // Stage 2: 对过滤后文档以日期为条件进行分组并计算
   {
      $group:
      {
         _id: { $dateToString: { format: "%Y-%m-%d", date: "$date" } },
         totalOrderValue: { $sum: { $multiply: [ "$price", "$quantity" ] } },
         averageOrderQuantity: { $avg: "$quantity" }
      }
   },
   // Stage 3: 按照订单价值倒序排列文档
   {
      $sort: { totalOrderValue: -1 }
   }
 ] )
```

计算结果：

```
{ "_id" : "2030-01-13", "totalOrderValue" : 350, "averageOrderQuantity" : 10 }
```

统计集合中文档数量

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

根据指定的字段进行过滤，去掉重复的文档

```
db.collection.distinct()
```

聚合管道顺序优化

聚合管道在执行的过程中有一个优化的阶段，以提高性能。

```
 $addFields: {
    maxTime: { $max: "$times" },
    minTime: { $min: "$times" }
} },
{ $project: {
    _id: 1, name: 1, times: 1, maxTime: 1,
minTime: 1,
    avgTime: { $avg: ["$maxTime", "$minTime"] }
} },
{ $match: {
    name: "Joe Schmoe",
    maxTime: { $lt: 20 },
    minTime: { $gt: 5 },
    avgTime: { $gt: 7 }
} }
```

优化思路：优化器把$match阶段分成了4个独立的过滤器，尽可能把过滤器放在$project操作前面，优化后的聚合管道如下：

```
{ $match: { name: "Joe Schmoe" } },
{ $addFields: {
    maxTime: { $max: "$times" },
    minTime: { $min: "$times" }
} },
{ $match: { maxTime: { $lt: 20 }, minTime: {
$gt: 5 } } },
{ $project: {
    _id: 1, name: 1, times: 1, maxTime: 1,
minTime: 1,
    avgTime: { $avg: ["$maxTime", "$minTime"] }
} },
{ $match: { avgTime: { $gt: 7 } } }
```

再例如：

```
{ $sort: { age : -1 } },
{ $match: { status: 'A' } }
```

优化后：

```
{ $match: { status: 'A' } },
{ $sort: { age : -1 } }
```

限制事项

- 返回结果集不能超过16M字节
- 单个管道中的stage数量不能超过1000个（MongoDB 5.0）

实时效果反馈

1.下列聚合操作命令中有错的是哪个？

A $sum、$avg、$unset

B $min 、$push、$first

C $last、$group、$project

D $match、$limit、$max

答案

1=>A

MongoDB索引Index

索引概述

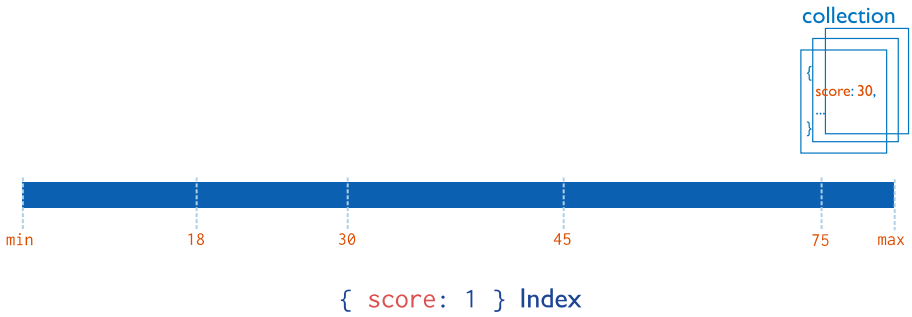
索引是一种单独的、物理的对数据库表中一列或多列的值进行排序的一种存储结构，它是某个表中一列或若干列值的集合和相应的指向表中物理标识这些值的数据页的逻辑指针清单。索引的作用相当于图书的目录，可以根据目录中的页码快速找到所需的内容。索引目标是提高数据库的查询效率，没有索引的话，查询会进行全表扫描（scan every document in a collection）,数据量大时严重降低了查询效率。默认情况下Mongo在一个集（collection）创建时，自动地对集合的_id创建了唯一索引。

索引类型

单键索引

MongoDB默认所有的集合在_id字段上有一个索引。

下图是一个在score字段上建立的升序/降序索引示意图。



创建一个名为records的集合，并插入下面的数据

```
{
  "_id": ObjectId("180c06a4ad577233f97tf825"),
  "score": 356,
  "location": { province: "Hebei", city:
"Tangshan" }
}
```

创建索引，1代表升序，-1代表降序

```
db.records.createIndex( { score: 1 } )
```

在内嵌字段上建立索引

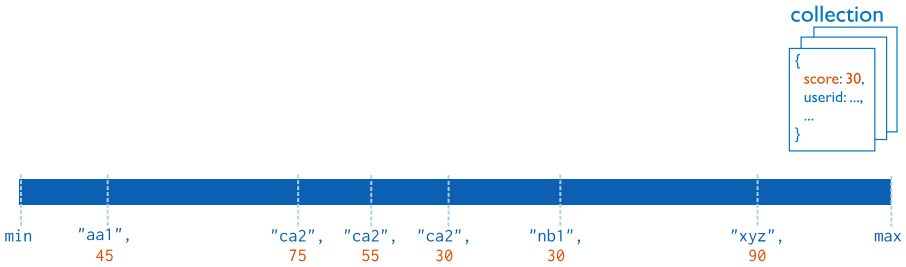
```
db.records.createIndex( { "location.province": 1 } )
```

在内嵌文档上建立索引

```
db.records.createIndex( { location: 1 } )
```

复合索引

复合索引是指单个索引结构指向多个字段，下图展示了在两个字段上建立索引



使用下面的数据来创建一个复合索引

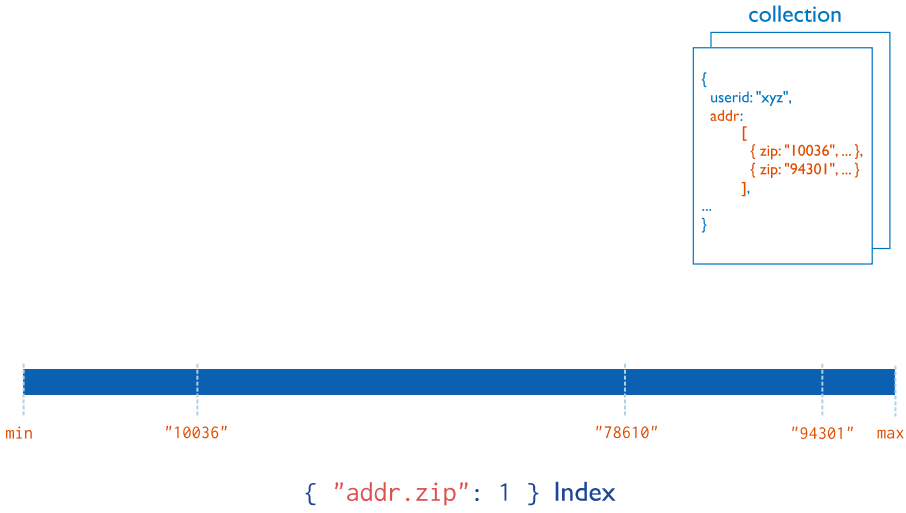
```
{
 "item": "Banana",
 "category": ["food", "produce", "grocery"],
 "location": "4th Street Store",
 "stock": 4,
 "type": "cases"
}
```

在item和stock字段上建立升序索引

```
db.products.createIndex( { "item": 1, "stock": 1 } )
```

多键索引

多键索引用于为数组中的元素创建索引



先创建集合inventory，使用下面的数据来创建多键索引

```
[{ _id: 5, type: "food", item: "aaa", ratings: [ 5, 8, 9 ] },
{ _id: 6, type: "food", item: "bbb", ratings: [ 5, 9 ] },
{ _id: 7, type: "food", item: "ccc", ratings: [ 9, 5, 8 ] },
{ _id: 8, type: "food", item: "ddd", ratings: [ 9, 5 ] },
{ _id: 9, type: "food", item: "eee", ratings: [ 5, 9, 5 ] }]
```

```
db.inventory.createIndex( { ratings: 1 } )
```

MongoDB使用多键索引查找在“ratings”数组中有“5”的文档，然后，MongoDB检索这些文档并筛选“ratings”数组等于查询数组“[5，9]”的文档。

```
db.inventory.find( { ratings: [ 5, 9 ] } )
```

地理空间索引

针对地理空间坐标数据创建索引。2dsphere索引用于存储和查找球面上的点，2d索引用于存储和查找平面上的点。

```
db.company.insert(
 {
  loc : { type: "Point", coordinates: [
116.502451, 40.014176 ] },
  name: "军博地铁",
  category : "Parks"
 }
)
```

```
db.company.createIndex( { loc : "2dsphere" } )
```

```
db.company.find({
 "loc" : {
   "$geoWithin" : {
    "$center":[[116.482451,39.914176],0.05]
   }
 }
})
```

```
db.places.insert({"name":"aa","addr":[32,32]})
db.places.insert({"name":"bb","addr":[30,22]})
db.places.insert({"name":"cc","addr":[28,21]})
db.places.insert({"name":"dd","addr":[34,26]})
db.places.insert({"name":"ee","addr":[34,27]})
db.places.insert({"name":"ff","addr":[39,28]})
```

```
db.places.find({})
db.places.createIndex({"addr":"2d"})
db.places.find({"addr":{"$within":{"$box":
[[0,0],[30,30]]}}})
```

全文索引

MongoDB提供了针对string内容的文本查询，Text Index支持任意属性值为string或string数组元素的索引查询。注意：一个集合仅支持最多一个Text Index，中文分词不理想推荐ES

```
db.fullText.insert({name:"aa",description:"no pains,no gains"})
db.fullText.insert({name:"ab",description:"pay pains,get gains"})
db.fullText.insert({name:"ac",description:"a friend in need,a friend in deed"})
```

创建索引并指定语言

```
db.fullText.createIndex(
   { description : "text" },
   { default_language: "english" }
)
```

```
db.fullText.find({"$text": {"$search": "pains"}})
```

全文索引名称

```
db.collection.createIndex(
   {
     content: "text",
     "users.comments": "text",
     "users.profiles": "text"
   }
)
```

生成的默认索引名：

```
content_text_users.comments_text_users.profiles_text
```

指定名称

```
db.collection.createIndex(
   {
     content: "text",
     "users.comments": "text",
     "users.profiles": "text"
   },
   {
     name: "MyTextIndex"
   }
)
```

使用指定名称删除索引

```
db.collection.dropIndex("MyTextIndex")
```

哈希索引

针对属性的哈希值进行索引查询，当要使用Hashed index时，MongoDB能够自动的计算hash值，无需程序计算hash值。注：hash index仅支持等于查询，不支持范围查询。

```
db.collection.createIndex({"field": "hashed"})
```

创建复合hash索引，4.4以后的版本

```
db.collection.createIndex( { "fieldA" : 1, "fieldB" : "hashed", "fieldC" : -1 } )
```

索引管理

获取针对某个集合的索引

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

索引的大小

```
db.collection.totalIndexSize()
```

索引的重建

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

索引的删除，注意： _id 对应的索引是删除不了的

```
db.collection.dropIndex("INDEX-NAME")
db.collection.dropIndexes()
```

1.MongoDB的索引种类有?

A 单键索引

B 多键索引

C 文本索引

D 以上者是

答案

1=>D

MongoDB应用实战

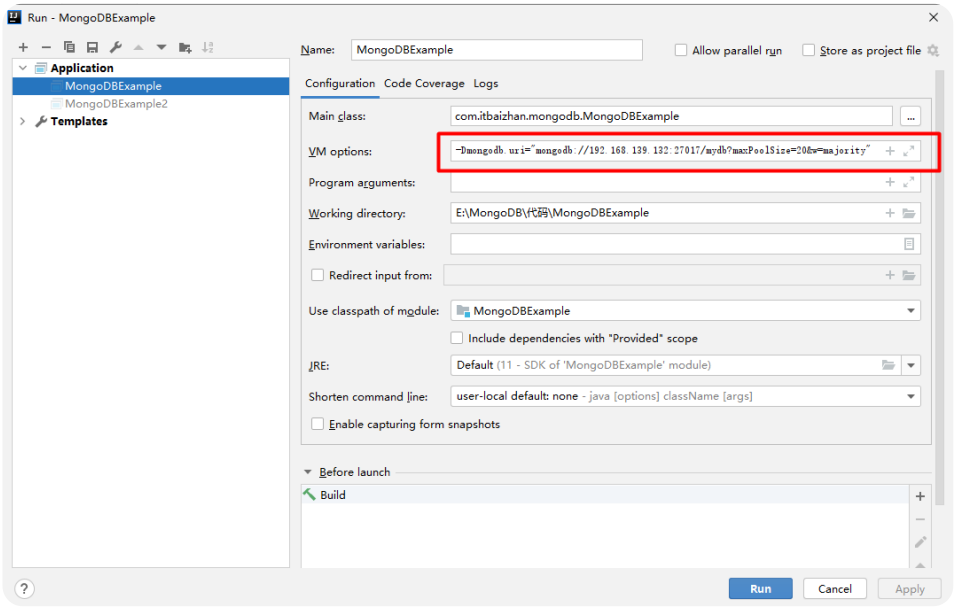
Java访问MongoDB

- 连接MongoDB

```
<dependency>
  <groupId>org.mongodb</groupId>
  <artifactId>mongo-java-driver</artifactId>
  <version>3.12.11</version>
</dependency>
```

```
import org.bson.BsonDocument;
import org.bson.BsonInt64;
import org.bson.Document;
import org.bson.conversions.Bson;
import com.mongodb.MongoClientSettings;
import com.mongodb.MongoException;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoDatabase;

public class MongoClientConnectionExample {
    public static void main(String[] args) {
        String connectionString =
System.getProperty("mongodb.uri");
        try (MongoClient mongoClient =
MongoClients.create(connectionString)) {
            List<Document> databases =
mongoClient.listDatabases().into(new
ArrayList<>());
```

```
                databases.forEach(db ->
System.out.println(db.toJson()));
            }
        }
}
```



- 创建集合

```
database.createCollection("exampleCollection");
```

- 文档添加

```
MongoClient mongoClient = new
MongoClient("192.168.139.132", 37017);
MongoDatabase database =
mongoClient.getDatabase("mydb");
MongoCollection<Document> collection =
database.getCollection("exampleCollection");
Document document = Document.parse(
"{name:'lisi',city:'bj',birth_day:new
ISODate('2001-08-
01'),expectSalary:18000}");
collection.insertOne(document );
mongoClient.close();
```

- 文档查询

```
MongoClient mongoClient = new
MongoClient("192.168.139.132", 27017);
MongoDatabase database =
mongoClient.getDatabase("mydb");
MongoCollection<Document> collection =
database.getCollection("exampleCollection");
Document sdoc=new Document();
//按expectSalary倒序
sdoc.append("expectSalary", -1);
FindIterable<Document> findIterable =
collection.find().sort(sdoc);
for (Document document : findIterable) {
    System.out.println(document);
```

```
}
 //按指定的属性值查询
 FindIterable<Document> documents =
collection.find(new Document("expectSalary",
new Document("$eq", 12000)));
for (Document document4 : documents) {
    System.out.println(document4);
}
mongoClient.close();
```

- 文档查询过滤

```
MongoClient mongoClient = new
MongoClient("192.168.139.132", 27017);
MongoDatabase database =
mongoClient.getDatabase("mydb");
MongoCollection<Document> collection =
database.getCollection("exampleCollection");
Document sdoc=new Document();
//按expectSalary倒序
sdoc.append("expectSalary", -1);
FindIterable<Document> findIterable =
collection.find(Filters.gt("expectSalary",21000
)).sort(sdoc);
for (Document document : findIterable) {
    System.out.println(document);
}
mongoClient.close();
```

1.Java直连MongoDB需引入哪个包

A spring-data-mongodb

B spring-boot-starter-mongodb

C mongo-java-driver

D mysql-connector-java

答案

1=>C

Springboot访问MongoDB

springboot访问MongoDb So easy！

MongoTemplate方式

- 引入依赖

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-
mongodb</artifactId>
  <version>2.6.8</version>
</dependency>
```

- 配置文件application.properties

```
spring.data.mongodb.host=192.168.139.132
spring.data.mongodb.port=27017
spring.data.mongodb.database=mydb
```

- DAO 实现类注入MongoTemplate 完成增删改查

```
@Autowired
protected MongoTemplate mongoTemplate;
```

MongoRepository 的方式

引入依赖，同MongoTemplate方式

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-
mongodb</artifactId>
  <version>2.6.8</version>
</dependency>
```

配置文件application.properties，同MongoTemplate方式

```
spring.data.mongodb.host=192.168.139.132
spring.data.mongodb.port=27017
spring.data.mongodb.database=mydb
```

编写 Repository接口继承MongoRepository

```
public interface OrdersRepository extends
MongoRepository<Orders, String> {
  List<Orders> findOrderByName(String name);
}
```

1.Springboot使用MongoDB采用哪种方式

A 注入MongoTemplate

B 编写接口继承MongoRepository

C 以上都是

答案

1=>C

MongoDB安全认证

安全认证概述

MongoDB 默认是没有账号的，可以直接连接，无须身份验证。实际项目中肯定是要权限验证的，否则后果不堪设想。从2016年开始发生了多起MongoDB黑客赎金事件，大部分MongoDB安全问题暴露出的短板其实是用户的安全意识不足，首先用户对于数据库的安全不重视，其次用户在使用过程中可能没有养成定期备份的好习

惯，最后是企业可能缺乏有经验和技术的专业人员。所以对MongoDB进行安全认证是必须要做的。

用户管理和安全认证

以auth方式启动MongoDB

```
docker run -itd --name mongo5 -p 27017:27017
mongo:xxx --auth
```

备份数据

```
mongodump -h 127.0.0.1:27017 -d mydb -o
/usr/local
```

恢复数据（在用户认证之后）

```
mongorestore -h localhost -u root -p 123456 --
db mydb /dump/mydb --authenticationDatabase
admin
```

进入容器内的mongo终端，切换到admin库添加用户，修改密码，验证用户以及删除用户

用户相关操作

用于创建 MongoDB 登录用户以及分配权限的方法

```
use admin;
db.createUser(
{
      user: "账号",
      pwd: "密码",
      roles: [
      { role: "角色", db: "安全认证的数据库" },
      { role: "角色", db: "安全认证的数据库" }
      ]
    }
)
```

user：创建的用户名称，如 admin、root 、zhangsan
pwd：用户登录的密码
roles：为用户分配的角色，不同的角色拥有不同的权限，参数是数组，可以同时设置多个
role：角色，MongoDB 已经约定好的角色，不同的角色对应不同的权限 后面会对role做详细解释
db：数据库实例名称，如 MongoDB 默认自带的有 admin、local、config、test 等，即为哪个数据库实例设置用户

例如：

```
db.createUser(
    {
        user:"root",
        pwd:"123321",
        roles:[{role:"root",db:"admin"}]
    }
)
```

修改密码

```
db.changeUserPassword( 'root' , '123456' );
```

添加角色

```
db.grantRolesToUser('用户名',[{ role:'角色名',
db:'数据库名'}])
```

验证用户

db.auth("账号","密码")，返回 1 说明认证成功

删除用户

db.dropUser("用户名")

1.MongoDB认证用户命令是

A db.createUser('username','password')

B db.auth('username','password')

C db.grantRolesToUser('username,[{ role:'rolename', db:'database'}])

D 以上都不对

答案

1=>B

MongoDB内置角色

角色	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	可以读取任何数据库并对库进行清理、修改、压缩，获取统计信息、执行检查等操作
userAdmin	可以在指定数据库里创建、删除和管理用户
readAnyDatabase	可以读取任何数据库中的数据，除了数据库config和local之外
readwriteAnyDatabase	可以读写任何数据库中的数据，除了数据库config和local之外
userAdminAnyDatabase	可以在指定的数据库中创建和修改用户，除了数据库config和local之外
dbAdminAnyDatabase	可以读取任何数据库并对库进行清理、修改、压缩，获取统计信息、执行检查等操作，除了数据库config和local之外
root	超级账号，超级权限
backup	备份数据权限
restore	从备份中恢复数据的权限

各个类型用户对应的角色

角色类型	角色名
数据库用户	read、readwrite
数据库管理角色	dbAdmin、userAdmin
所有数据库角色	readAnyDatabase、readWriteAnyDatabase、userAdminAnyDatabase、dbAdminAnyDatabase
备份恢复角色	backup、restore
超级用户角色	root

1.MongoDB内置角色不包括哪个?

A backup

B restore

C root

D admin

答案

1=>D

基于角色的访问控制

- 创建管理员

MongoDB 服务端开启安全检查之前，至少需要有一个管理员账号，admin 数据库中的用户都被视为管理员如果 admin 库没有任何用户的话，即使在其他数据库中创建了用户，启用身份验证，默认的连接方式依然会有超级权限，即仍然可以不验证账号密码照样能进行 CRUD，安全认证相当于无效。

```
>use admin
switched to db admin
> db
admin
> db.createUser(
... {
... user:"root",
... pwd:"123456",
... roles:[{role:"root",db:"admin"}]
... })
```

- 创建普通用户

创建 mydb数据库并创建两个用户，zhangsan 拥有读写权限，lisi 拥有只读权限测试这两个账户的权限。以超级管理员登录测试权限。

```
> use mydb
switched to db mydb
>  db.c1.insert({name:"testdb1"})
WriteResult({ "nInserted" : 1 })
> db.c2.insert({name:"testdb1"})
WriteResult({ "nInserted" : 1 })
> show tables
c1
c2
> db.c1.find()
{ "_id" : ObjectId("62a00e5c1eb2c6ab85dd5eec"),
"name" : "testdb1" }
> db.c1.find({})
{ "_id" : ObjectId("62a00e5c1eb2c6ab85dd5eec"),
"name" : "testdb1" }
> show dbs
admin      0.000GB
config     0.000GB
local      0.000GB
mydb       0.001GB
>
```

如下图所示，为 mydb数据库创建了两个用户，zhangsan 拥有读写权限，lisi 拥有只读权限，密码都是 123456。

```
use mydb
switched to db mydb
> db
mydb
```

```
> db.createUser({
... user:"zhangsan",
... pwd:"123456",
... roles:[{role:"readWrite",db:"mydb"}]
... })
> db.createUser({
... user:"lisi",
... pwd:"123456",
... roles:[{role:"read",db:"mydb"}]
... })
```

- 以普通用户登录验证权限

普通用户现在仍然像以前一样进行登录，如下所示直接登录进入mydb数据库中，登录是成功的，只是登录后日志少了很多东西，而且执行 show dbs 命令，以及 show tables 等命令都是失败的，即使没有被安全认证的数据库，用户同样操作不了，这都是因为权限不足，一句话：用户只能在自己权限范围内的数据库中进行操作。

```
> db.auth("zhangsan","123456")
1
> show dbs
mydb 0.001GB
> show tables
c1
c2
```

- 以管理员登录验证权限

客户端管理员以root用户登录，安全认证通过后，拥有对所有数据库的所有权限

```
> use admin
switched to db admin
> db.auth("root","123456")
1
> show dbs
...
```

1.以安全认证方式运行MongoDB下列说法错误的是?

A 需要在admin库中添加用户

B 需要在mongod.conf配置文件中设置auth=true

C 如果是docker启动需要在docker run命令后添加参数--auth

D 必须在admin库中创建root角色用户

答案

1=>D

2.MongoDB查看所有创建的用户命令是?

A db.getUsers()

B db.findAllUsers()

C db.system.uses.find().pretty()

D use admin

答案

2=>C