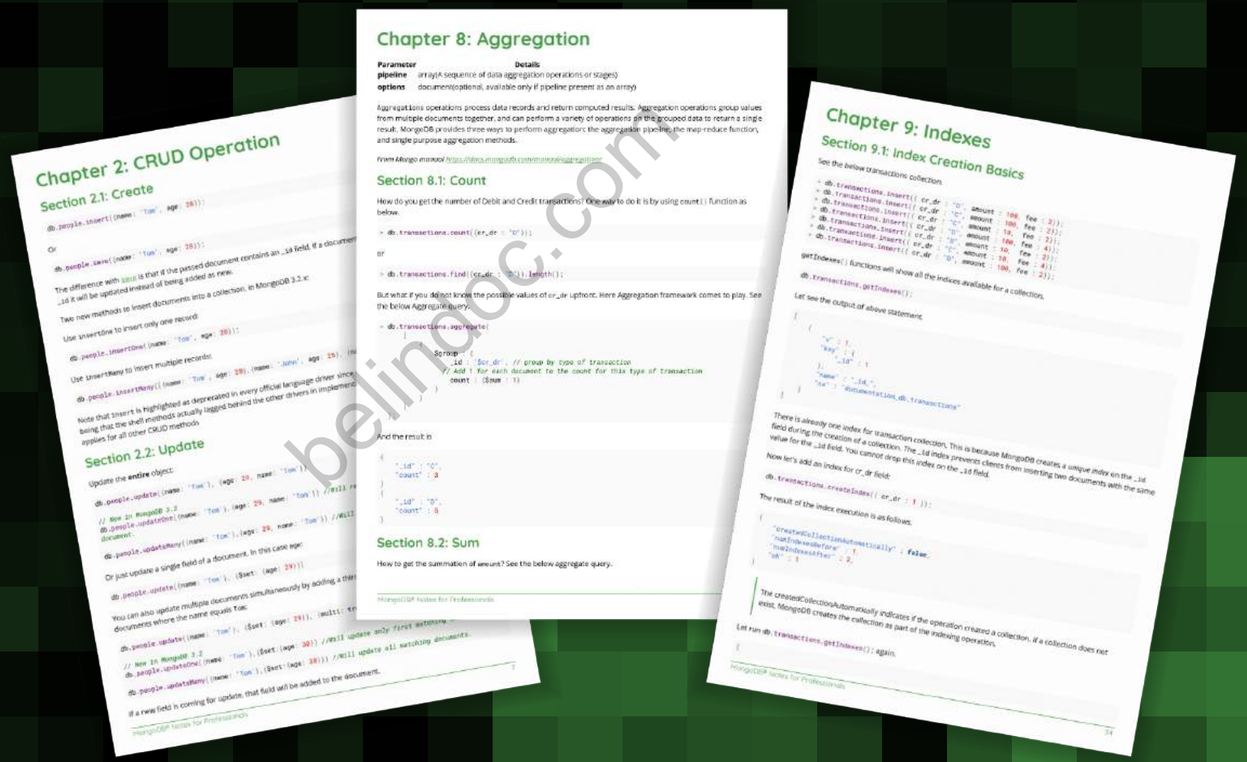
专业人员笔记

MongoDB®

专业人员笔记

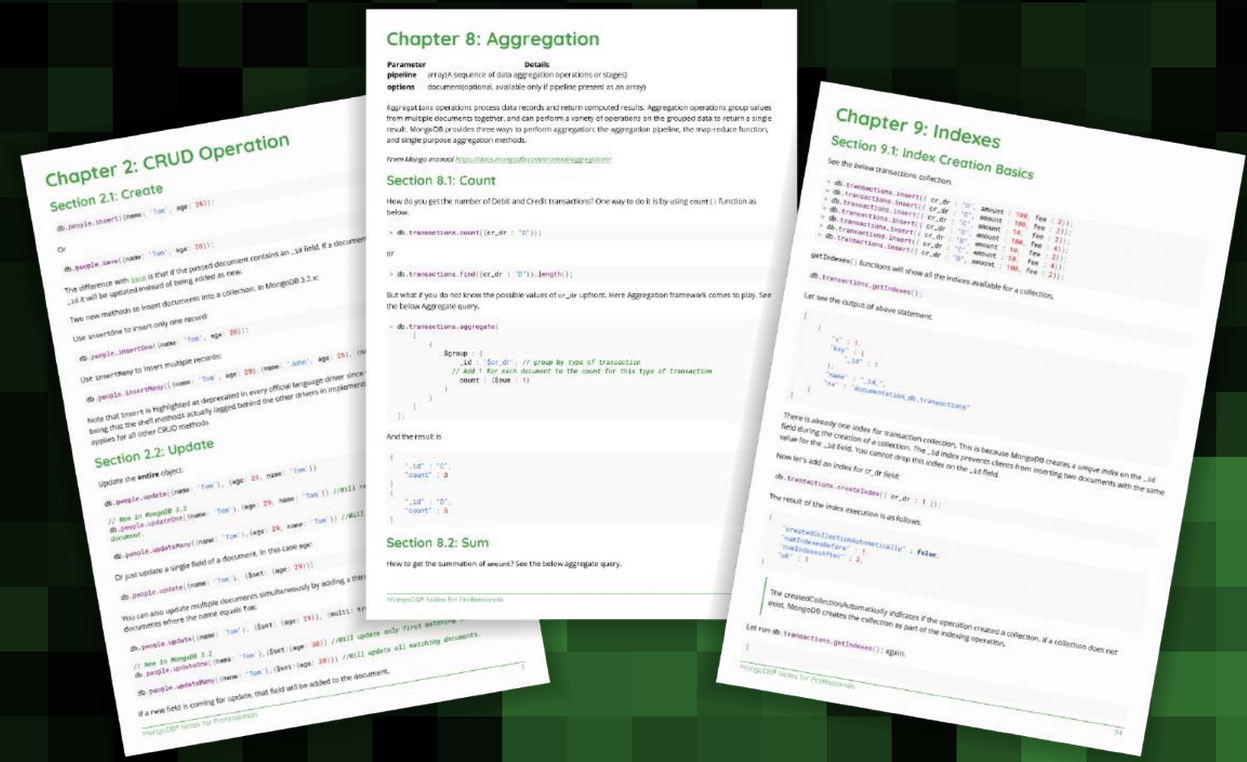


60+ 页

专业提示和技巧

MongoDB®

Notes for Professionals



60+ pages

of professional hints and tricks

目录

Contents

关于

欢迎随意免费分享此PDF，
本书的最新版本可从以下网址下载：
https://goalkicker.com/MongoDBBook

本书中提供的信息不保证正确或准确，使用风险自负

请将反馈和更正发送至web@petercv.com

About

Please feel free to share this PDF with anyone for free,
latest version of this book can be downloaded from:
https://goalkicker.com/MongoDBBook

The information presented in this book is not guaranteed to be correct nor accurate, use at your own risk

Please send feedback and corrections to web@petercv.com

第1章：MongoDB入门

版本发布日期

3.6.1	2017-12-26
3.4	2016-11-29
3.2	2015-12-08
3.0	2015-03-03
2.6	2014-04-08
2.4	2013-03-19
2.2	2012-08-29
2.0	2011-09-12
1.8	2011-03-16
1.6	2010-08-31
1.4	2010-03-25
1.2	2009-12-10

第1.1节：在MongoDB中执行JavaScript文件

```
./mongo localhost:27017/mydb myjsfile.js
```

说明： 此操作在连接到通过localhost接口、端口为27017的mongod实例上的mydb数据库的mongo shell中执行myjsfile.js脚本。localhost:27017不是必须的，因为这是mongodb使用的默认端口。

此外，您也可以在mongo控制台内运行.js文件。

```
>load("myjsfile.js")
```

第1.2节：使find的输出在shell中可读

我们向集合test中添加三条记录，如下：

```
> db.test.insert({"key":"value1","key2":"Val2","key3":"val3"})
WriteResult({ "nInserted" : 1 })
> db.test.insert({"key":"value2","key2":"Val21","key3":"val31"})
WriteResult({ "nInserted" : 1 })
> db.test.insert({"key":"value3","key2":"Val22","key3":"val33"})
WriteResult({ "nInserted" : 1 })
```

如果通过find查看它们，显示会非常难看。

```
> db.test.find()
{ "_id" : ObjectId("5790c5cecae25b3d38c3c7ae"), "key" : "value1", "key2" : "Val2
", "key3" : "val3" }
{ "_id" : ObjectId("5790c5d9cae25b3d38c3c7af"), "key" : "value2", "key2" : "Val2
1", "key3" : "val31" }
{ "_id" : ObjectId("5790c5e9cae25b3d38c3c7b0"), "key" : "value3", "key2" : "Val2
2", "key3" : "val33" }
```

为了解决这个问题并使它们可读，使用pretty()函数。

```
> db.test.find().pretty()
```


Chapter 1: Getting started with MongoDB

Version Release Date

3.6.1	2017-12-26
3.4	2016-11-29
3.2	2015-12-08
3.0	2015-03-03
2.6	2014-04-08
2.4	2013-03-19
2.2	2012-08-29
2.0	2011-09-12
1.8	2011-03-16
1.6	2010-08-31
1.4	2010-03-25
1.2	2009-12-10

Section 1.1: Execution of a JavaScript file in MongoDB

```
./mongo localhost:27017/mydb myjsfile.js
```

Explanation: This operation executes the `myjsfile.js` script in a `mongo` shell that connects to the `mydb` database on the `mongod` instance accessible via the `localhost` interface on port `27017`. `localhost:27017` is not mandatory as this is the default port `mongodb` uses.

Also, you can run a `.js` file from within `mongo` console.

```
>load("myjsfile.js")
```

Section 1.2: Making the output of find readable in shell

We add three records to our collection test as:

```
> db.test.insert({"key":"value1","key2":"Val2","key3":"val3"})
WriteResult({ "nInserted" : 1 })
> db.test.insert({"key":"value2","key2":"Val21","key3":"val31"})
WriteResult({ "nInserted" : 1 })
> db.test.insert({"key":"value3","key2":"Val22","key3":"val33"})
WriteResult({ "nInserted" : 1 })
```

If we see them via find, they will look very ugly.

```
> db.test.find()
{ "_id" : ObjectId("5790c5cecae25b3d38c3c7ae"), "key" : "value1", "key2" : "Val2
", "key3" : "val3" }
{ "_id" : ObjectId("5790c5d9cae25b3d38c3c7af"), "key" : "value2", "key2" : "Val2
1", "key3" : "val31" }
{ "_id" : ObjectId("5790c5e9cae25b3d38c3c7b0"), "key" : "value3", "key2" : "Val2
2", "key3" : "val33" }
```

To work around this and make them readable, use the **pretty**() function.

```
> db.test.find().pretty()
```

```
{
        "_id" : ObjectId("5790c5cecae25b3d38c3c7ae"),
        "key" : "value1",
        "key2" : "Val2",
        "key3" : "val3"
}
{
        "_id" : ObjectId("5790c5d9cae25b3d38c3c7af"),
        "key" : "value2",
        "key2" : "Val21",
        "key3" : "val31"
}
{
        "_id" : ObjectId("5790c5e9cae25b3d38c3c7b0"),
        "key" : "value3",
        "key2" : "Val22",
        "key3" : "val33"
}
>
```

第1.3节：补充术语

SQL术语	MongoDB术语
数据库	数据库
表	集合
实体 / 行	文档
列	关键 / 字段
表连接	嵌入式文档
主键	主键（默认键_id由mongodb自身提供）

第1.4节：安装

安装MongoDB，请按照以下步骤操作：

- **对于Mac OS：**
  - Mac OS有两种选择：手动安装或homebrew。
  - **使用homebrew安装：**
    - 在终端输入以下命令：

      ```
      $ brew install mongodb
      ```
  - **手动安装：**
    - 下载最新版本 here。确保下载了合适的文件，特别要检查你的操作系统类型是32位还是64位。下载的文件格式为gz。
    - 进入下载该文件的目录。然后输入以下命令：

      ```
      $ tar xvf mongodb-osx-xyz.tgz
      ```

      “xyz”处会有版本和系统类型信息。解压后的文件夹名称与gz文件相同。文件夹内有一个名为bin的子文件夹，里面包含多个二进制文件，包括mongod和mongo。
    - 默认情况下，服务器将数据保存在/data/db文件夹中。因此，我们需要创建该目录，然后


```
{
        "_id" : ObjectId("5790c5cecae25b3d38c3c7ae"),
        "key" : "value1",
        "key2" : "Val2",
        "key3" : "val3"
}
{
        "_id" : ObjectId("5790c5d9cae25b3d38c3c7af"),
        "key" : "value2",
        "key2" : "Val21",
        "key3" : "val31"
}
{
        "_id" : ObjectId("5790c5e9cae25b3d38c3c7b0"),
        "key" : "value3",
        "key2" : "Val22",
        "key3" : "val33"
}
>
```

Section 1.3: Complementary Terms

SQL Terms	MongoDB Terms
Database	Database
Table	Collection
Entity / Row	Document
Column	Key / Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key `_id` provided by mongodb itself)

Section 1.4: Installation

To install MongoDB, follow the steps below:

- **For Mac OS:**
  - There are two options for Mac OS: manual install or homebrew.
  - **Installing with homebrew:**
    - Type the following command into the terminal:

      ```
      $ brew install mongodb
      ```
  - **Installing manually:**
    - Download the latest release here. Make sure that you are downloading the appropriate file, specially check whether your operating system type is 32-bit or 64-bit. The downloaded file is in format `tgz`.
    - Go to the directory where this file is downloaded. Then type the following command:

      ```
      $ tar xvf mongodb-osx-xyz.tgz
      ```

      Instead of `xyz`, there would be some version and system type information. The extracted folder would be same name as the `tgz` file. Inside the folder, their would be a subfolder named `bin` which would contain several binary file along with `mongod` and `mongo`.
    - By default server keeps data in folder `/data/db`. So, we have to create that directory and then

使用以下命令启动服务器：

```
$ sudo bash
# mkdir -p /data/db
# chmod 777 /data
# chmod 777 /data/db
# exit
```

- 要启动服务器，应在当前目录下执行以下命令：

```
$ ./mongod
```

默认情况下，它会在27017端口启动服务器。

- 要启动客户端，应打开一个新的终端，且目录与之前相同。然后
以下命令将启动客户端并连接到服务器。

```
$ ./mongo
```

默认情况下，它连接到 test数据库。如果你看到类似connecting to: test的行，说明你已经成功安装了MongoDB。恭喜！现在，你可以测试Hello World以增强信心。

- **对于Windows：**
  - 在这里下载最新版本。确保下载了合适的文件，特别是
检查你的操作系统类型是32位还是64位。
  - 下载的二进制文件扩展名为exe。运行它。它会弹出安装向导。
  - 点击Next。
  - 接受许可协议并点击Next。
  - 选择Complete安装。
  - 点击安装。可能会弹出一个窗口，要求管理员权限。点击是。
  - 安装完成后点击完成。
  - 现在，mongodb 已安装在路径C:/Program Files/MongoDB/Server/3.2/bin。你的版本可能不是3.2，路径名称会相应更改。
  - bin目录包含多个二进制文件，包括mongod和mongo。若要从其他文件夹运行，可以将该路径添加到系统路径。操作步骤如下：
    - 右键点击我的电脑，选择属性。
    - 点击左侧窗格的高级系统设置。
    - **在高级选项卡下点击环境变量...。**
    - 在系统变量部分选择Path，点击编辑...。
    - 在Windows 10之前，添加分号后粘贴上述路径。Windows 10及以后版本，有一个新建按钮用于添加新路径。
    - 点击确定保存更改。
  - 现在，在你想运行服务器的位置创建一个名为data的文件夹，里面有一个名为db的子文件夹。
  - 从那里启动命令提示符。可以在cmd中更改路径，或者点击“在此处打开命令窗口”，该选项在文件夹GUI空白处右键点击后按住Shift和Ctrl键同时按下时可见。

run the server having the following commands:

```
$ sudo bash
# mkdir -p /data/db
# chmod 777 /data
# chmod 777 /data/db
# exit
```

- To start the server, the following command should be given from the current location:

```
$ ./mongod
```

It would start the server on port 27017 by default.

- To start the client, a new terminal should be opened having the same directory as before. Then the following command would start the client and connect to the server.

```
$ ./mongo
```

By default it connects to the `test` database. If you see the line like `connecting to: test`. Then you have successfully installed MongoDB. Congrats! Now, you can test Hello World to be more confident.

- **For Windows:**
  - Download the latest release here. Make sure that you are downloading the appropriate file, specially check whether your operating system type is 32-bit or 64-bit.
  - The downloaded binary file has extension `exe`. Run it. It will prompt an installation wizard.
  - Click **Next**.
  - **Accept** the licence agreement and click **Next**.
  - Select **Complete** Installation.
  - Click on **Install**. It might prompt a window for asking administrator's permission. Click **Yes**.
  - After installation click on **Finish**.
  - Now, the mongodb is installed on the path `C:/Program Files/MongoDB/Server/3.2/bin`. Instead of version 3.2, there could be some other version for your case. The path name would be changed accordingly.
  - `bin` directory contain several binary file along with `mongod` and `mongo`. To run it from other folder, you could add the path in system path. To do it:
    - Right click on **My Computer** and select **Properties**.
    - Click on **Advanced system setting** on the left pane.
    - Click on **Environment Variables...** under the **Advanced** tab.
    - Select **Path** from **System variables** section and click on **Edit...**.
    - Before Windows 10, append a semi-colon and paste the path given above. From Windows 10, there is a **New** button to add new path.
    - Click **OK**s to save changes.
  - Now, create a folder named `data` having a sub-folder named `db` where you want to run the server.
  - Start command prompt from their. Either changing the path in cmd or clicking on **Open command window here** which would be visible after right clicking on the empty space of the folder GUI pressing

同时按下Shift和Ctrl键。

- 输入命令以启动服务器：

```
> mongod
```

默认情况下，它会在27017端口启动服务器。

- 打开另一个命令提示符，输入以下命令启动客户端：

```
> mongo
```

- 默认情况下，它连接到test数据库。如果你看到类似connecting to: test的行，那么你已经成功安装了MongoDB。恭喜！现在，你可以测试Hello World以增强信心。

- 对于Linux：几乎与Mac OS相同，只是需要一些等效命令。

  - 对于基于Debian的发行版（使用apt-get）：
    - 导入 MongoDB 仓库密钥。

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927
gpg: Total number processed: 1\
gpg:               imported: 1  (RSA: 1)
```

    - 在 Ubuntu 16.04 上将仓库添加到软件包列表。

```
$ echo "deb http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.2 multiverse"
| sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list
```

    - 在 Ubuntu 14.04 上。

```
$ echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.2 multiverse"
| sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list
```

    - 更新软件包列表。

```
$ sudo apt-get update
```

    - 安装 MongoDB。

```
$ sudo apt-get install mongodb-org
```

  - 对于基于Red Hat的发行版（使用yum）：
    - 使用你喜欢的文本编辑器。

```
$ vi /etc/yum.repos.d/mongodb-org-3.4.repo
```

    - 粘贴以下内容。

```
[mongodb-org-3.4]
name=MongoDB 仓库
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/3.4/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-3.4.asc
```

    - 更新软件包列表。

```
$ sudo yum update
```

    - 安装 MongoDB

```
$ sudo yum install mongodb-org
```


the Shift and Ctrl key together.

- Write the command to start the server:

```
> mongod
```

It would start the server on port 27017 by default.

- Open another command prompt and type the following to start client:

```
> mongo
```

- By default it connects to the `test` database. If you see the line like `connecting to: test`. Then you have successfully installed MongoDB. Congrats! Now, you can test Hello World to be more confident.

- **For Linux:** Almost same as Mac OS except some equivalent command is needed.

  - For Debian-based distros (using `apt-get`):
    - Import MongoDB Repository key.

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927
gpg: Total number processed: 1\
gpg:               imported: 1  (RSA: 1)
```

    - Add repository to package list on **Ubuntu 16.04**.

```
$ echo "deb http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.2 multiverse"
| sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list
```

    - on **Ubuntu 14.04**.

```
$ echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.2 multiverse"
| sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list
```

    - Update package list.

```
$ sudo apt-get update
```

    - Install MongoDB.

```
$ sudo apt-get install mongodb-org
```

  - For Red Hat based distros (using `yum`):
    - use a text editor which you prefer.

```
$ vi /etc/yum.repos.d/mongodb-org-3.4.repo
```

    - Paste following text.

```
[mongodb-org-3.4]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/3.4/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-3.4.asc
```

    - Update package list.

```
$ sudo yum update
```

    - Install MongoDB

```
$ sudo yum install mongodb-org
```

第1.5节：mongo shell上的基本命令

显示所有可用的数据库：

```
show dbs;
```

选择一个特定的数据库进行访问，例如mydb。如果mydb不存在，将会创建它：

```
use mydb;
```

显示数据库中的所有集合（请先选择一个数据库，见上文）：

```
show collections;
```

显示所有可用于数据库的函数：

```
db.mydb.help();
```

要检查您当前选择的数据库，请使用命令 db

```
> db
mydb
```

db.dropDatabase() 命令用于删除一个已存在的数据库。

```
db.dropDatabase()
```

第1.6节：Hello World

安装过程完成后，应在mongo shell（客户端终端）中输入以下命令行。

```
> db.world.insert({ "speech" : "Hello World!" });
> cur = db.world.find();x=cur.next();print(x["speech"]);
```

> Hello World!

说明：

- 在第一行中，我们在默认数据库 test 和名为 world 的集合中插入了一个 { 键 : 值 } 对文档。
- 在第二行中，我们检索刚刚插入的数据。检索到的数据保存在一个名为cur的JavaScript变量中。然后通过next()函数，我们检索到第一个也是唯一的文档，并将其保存在另一个名为x的JavaScript变量中。随后通过提供键值打印该文档的值。

Section 1.5: Basic commands on mongo shell

Show all available databases:

```
show dbs;
```

Select a particular database to access, e.g. `mydb`. This will create `mydb` if it does not already exist:

```
use mydb;
```

Show all collections in the database (be sure to select one first, see above):

```
show collections;
```

Show all functions that can be used with the database:

```
db.mydb.help();
```

To check your currently selected database, use the command `db`

```
> db
mydb
```

`db.dropDatabase()` command is used to drop a existing database.

```
db.dropDatabase()
```

Section 1.6: Hello World

After installation process, the following lines should be entered in mongo shell (client terminal).

```
> db.world.insert({ "speech" : "Hello World!" });
> cur = db.world.find();x=cur.next();print(x["speech"]);
```

> Hello World!

Explanation:

- In the first line, we have inserted a `{ key : value }` paired document in the default database `test` and in the collection named `world`.
- In the second line we retrieve the data we have just inserted. The retrieved data is kept in a javascript variable named `cur`. Then by the `next()` function, we retrieved the first and only document and kept it in another js variable named `x`. Then printed the value of the document providing the key.

第二章：CRUD操作

第2.1节：创建

```
db.people.insert({name: '汤姆', age: 28});
```

或者

```
db.people.save({name: '汤姆', age: 28});
```

save与save的区别在于，如果传入的文档包含_id字段，且已有文档存在该_id，则该文档会被更新，而不是作为新文档添加。

在MongoDB 3.2.x中，插入文档到集合的两种新方法：

使用insertOne来插入单条记录：

```
db.people.insertOne({name: 'Tom', age: 28});
```

使用insertMany插入多条记录：

```
db.people.insertMany([{name: 'Tom', age: 28},{name: 'John', age: 25}, {name: 'Kathy', age: 23}])
```

注意，自3.0版本起，所有官方语言驱动中`insert`均被标记为已弃用。完整的区别
在于shell方法实际上在实现该方法方面落后于其他驱动。相同情况
也适用于所有其他CRUD方法。

第2.2节：更新

更新整个对象：

```
db.people.update({name: 'Tom'}, {age: 29, name: 'Tom'})

// MongoDB 3.2中新增加
db.people.updateOne({name: 'Tom'},{age: 29, name: 'Tom'}) //只会替换第一个匹配的文档。


db.people.updateMany({name: 'Tom'},{age: 29, name: 'Tom'}) //会替换所有匹配的文档。
```

或者只更新文档的单个字段。在此例中为age：

```
db.people.update({name: 'Tom'}, {$set: {age: 29}})
```

您也可以通过添加第三个参数同时更新多个文档。此查询将更新所有名称等于Tom的文档：

```
db.people.update({name: 'Tom'}, {$set: {age: 29}}, {multi: true})

// MongoDB 3.2中新增加
db.people.updateOne({name: 'Tom'},{$set:{age: 30}) //只会更新第一个匹配的文档。

db.people.updateMany({name: 'Tom'},{$set:{age: 30}}) //会更新所有匹配的文档。
```

如果有新的字段需要更新，该字段将被添加到文档中。

Chapter 2: CRUD Operation

Section 2.1: Create

```
db.people.insert({name: 'Tom', age: 28});
```

Or

```
db.people.save({name: 'Tom', age: 28});
```

The difference with `save` is that if the passed document contains an `_id` field, if a document already exists with that `_id` it will be updated instead of being added as new.

Two new methods to insert documents into a collection, in MongoDB 3.2.x:

Use `insertOne` to insert only one record:

```
db.people.insertOne({name: 'Tom', age: 28});
```

Use `insertMany` to insert multiple records:

```
db.people.insertMany([{name: 'Tom', age: 28},{name: 'John', age: 25}, {name: 'Kathy', age: 23}])
```

Note that `insert` is highlighted as deprecated in every official language driver since version 3.0. The full distinction being that the shell methods actually lagged behind the other drivers in implementing the method. The same thing applies for all other CRUD methods

Section 2.2: Update

Update the **entire** object:

```
db.people.update({name: 'Tom'}, {age: 29, name: 'Tom'})

// New in MongoDB 3.2
db.people.updateOne({name: 'Tom'},{age: 29, name: 'Tom'}) //Will replace only first matching
document.

db.people.updateMany({name: 'Tom'},{age: 29, name: 'Tom'}) //Will replace all matching documents.
```

Or just update a single field of a document. In this case `age`:

```
db.people.update({name: 'Tom'}, {$set: {age: 29}})
```

You can also update multiple documents simultaneously by adding a third parameter. This query will update all documents where the name equals `Tom`:

```
db.people.update({name: 'Tom'}, {$set: {age: 29}}, {multi: true})

// New in MongoDB 3.2
db.people.updateOne({name: 'Tom'},{$set:{age: 30}) //Will update only first matching document.

db.people.updateMany({name: 'Tom'},{$set:{age: 30}}) //Will update all matching documents.
```

If a new field is coming for update, that field will be added to the document.

```
db.people.updateMany({name: 'Tom'},{$set:{age: 30, salary:50000}})//文档中也会有`salary`字段。
```

如果需要替换文档，

```
db.collection.replaceOne({name:'Tom'}, {name:'Lakmal',age:25,address:'Sri Lanka'})
```

可以使用。

注意：用于识别对象的字段将保存在更新后的文档中。未在更新部分定义的字段将从文档中移除。

第2.3节：删除

删除所有匹配查询参数的文档：

```
// MongoDB 3.2中新增加
db.people.deleteMany({name: 'Tom'})

// 所有版本
db.people.remove({name: 'Tom'})
```

或者只删除一个

```
// MongoDB 3.2中新增加
db.people.deleteOne({name: 'Tom'})

// 所有版本
db.people.remove({name: 'Tom'}, true)
```

MongoDB的`remove()`方法。如果你执行此命令时不带任何参数或带空参数，它将删除集合中的所有文档。

```
db.people.remove();
```

或者

```
db.people.remove({});
```

第2.4节：读取

查询people集合中所有name字段值为'Tom'的文档：

```
db.people.find({name: 'Tom'})
```

或者只查询第一个：

```
db.people.findOne({name: 'Tom'})
```

你也可以通过传递字段选择参数来指定返回哪些字段。以下示例将排除`_id`字段，仅包含`age`字段：

```
db.people.find({name: 'Tom'}, {_id: 0, age: 1})
```



```
db.people.updateMany({name: 'Tom'},{$set:{age: 30, salary:50000}})// Document will have `salary` field as well.
```

If a document is needed to be replaced,

```
db.collection.replaceOne({name:'Tom'}, {name:'Lakmal',age:25,address:'Sri Lanka'})
```

can be used.

Note: Fields you use to identify the object will be saved in the updated document. Field that are not defined in the update section will be removed from the document.

Section 2.3: Delete

Deletes all documents matching the query parameter:

```
// New in MongoDB 3.2
db.people.deleteMany({name: 'Tom'})

// All versions
db.people.remove({name: 'Tom'})
```

Or just one

```
// New in MongoDB 3.2
db.people.deleteOne({name: 'Tom'})

// All versions
db.people.remove({name: 'Tom'}, true)
```

MongoDB's `remove()` method. If you execute this command without any argument or without empty argument it will remove all documents from the collection.

```
db.people.remove();
```

or

```
db.people.remove({});
```

Section 2.4: Read

Query for all the docs in the `people` collection that have a `name` field with a value of `'Tom'`:

```
db.people.find({name: 'Tom'})
```

Or just the first one:

```
db.people.findOne({name: 'Tom'})
```

You can also specify which fields to return by passing a field selection parameter. The following will exclude the `_id` field and only include the `age` field:

```
db.people.find({name: 'Tom'}, {_id: 0, age: 1})
```

注意：默认情况下，即使你不请求，_id字段也会被返回。如果你不想返回_id字段，可以按照前面的示例，通过指定_id: 0（或_id: false）来排除它。如果你想查找包含国家、城市等的地址对象这样的子记录，

```
db.people.find({'address.country': 'US'})
```

如果需要，也请指定字段

```
db.people.find({'address.country': 'US'}, {'name': true, 'address.city': true})请记住，
结果有一个`.pretty()`方法，可以美观地打印结果JSON：

db.people.find().pretty()
```

第2.5节：嵌入文档的更新

对于以下模式：

```
{name: '汤姆', age: 28, marks: [50, 60, 70]}
```

将汤姆的分数中50的更新为55（使用位置操作符$）：

```
db.people.update({name: "汤姆", marks: 50}, {"$set": {"marks.$": 55}})
```

对于以下模式：

```
{name: '汤姆', age: 28, marks: [{subject: "英语", marks: 90},{subject: "数学", marks: 100},
{subject: "计算机", marks: 20}]}
```

将汤姆的英语成绩更新为85：

```
db.people.update({name: "Tom", "marks.subject": "English"},{"$set":{"marks.$.marks": 85}})
```

解释上述示例：

通过使用{name: "Tom", "marks.subject": "English"}，你将获得marks数组中subject为English的对象的位置。在"marks.$.marks"中，$用于更新marks数组中该位置的元素

更新数组中的值

位置操作符$用于标识数组中要更新的元素，而无需明确指定该元素在数组中的位置。

考虑一个名为students的集合，包含以下文档：

```
{ "_id" : 1, "grades" : [ 80, 85, 90 ] }
{ "_id" : 2, "grades" : [ 88, 90, 92 ] }
{ "_id" : 3, "grades" : [ 85, 100, 90 ] }
```

如果你不知道元素在数组中的位置，想要将第一个文档中grades数组中的80更新为82，可以使用位置操作符$：

```
db.students.update(
   { _id: 1, grades: 80 },
   { $set: { "grades.$" : 82 } }
```


Note: by default, the `_id` field will be returned, even if you don't ask for it. If you would like not to get the `_id` back, you can just follow the previous example and ask for the `_id` to be excluded by specifying `_id: 0` (or `_id: false`).If you want to find sub record like address object contains country, city, etc.

```
db.people.find({'address.country': 'US'})
```

& specify field too if required

```
db.people.find({'address.country': 'US'}, {'name': true, 'address.city': true})Remember that the
result has a `.pretty()` method that pretty-prints resulting JSON:

db.people.find().pretty()
```

Section 2.5: Update of embedded documents

For the following schema:

```
{name: 'Tom', age: 28, marks: [50, 60, 70]}
```

Update Tom's marks to 55 where marks are 50 (Use the positional operator $):

```
db.people.update({name: "Tom", marks: 50}, {"$set": {"marks.$": 55}})
```

For the following schema:

```
{name: 'Tom', age: 28, marks: [{subject: "English", marks: 90},{subject: "Maths", marks: 100},
{subject: "Computes", marks: 20}]}
```

Update Tom's English marks to 85 :

```
db.people.update({name: "Tom", "marks.subject": "English"},{"$set":{"marks.$.marks": 85}})
```

Explaining above example:

By using {name: "Tom", "marks.subject": "English"} you will get the position of the object in the marks array, where subject is English. In "marks.$.marks", $ is used to update in that position of the marks array

Update Values in an Array

The positional $ operator identifies an element in an array to update without explicitly specifying the position of the element in the array.

Consider a collection students with the following documents:

```
{ "_id" : 1, "grades" : [ 80, 85, 90 ] }
{ "_id" : 2, "grades" : [ 88, 90, 92 ] }
{ "_id" : 3, "grades" : [ 85, 100, 90 ] }
```

To update 80 to 82 in the grades array in the first document, use the positional $ operator if you do not know the position of the element in the array:

```
db.students.update(
   { _id: 1, grades: 80 },
   { $set: { "grades.$" : 82 } }
```

```
)
```

第2.6节：更多更新算子

除了$set操作符之外，你还可以使用其他操作符来更新文档。 $push操作符允许你向数组中添加一个值，这里我们将向nicknames数组中添加一个新的昵称。

```
db.people.update({name: 'Tom'}, {$push: {nicknames: 'Tommy'}})
// 这会将字符串'Tommy'添加到Tom文档中的nicknames数组中。
```

$pull操作符与$push相反，你可以从数组中移除特定的项。

```
db.people.update({name: 'Tom'}, {$pull: {nicknames: 'Tommy'}})
// 这会从Tom文档中的nicknames数组中移除字符串'Tommy'。
```

$pop操作符允许你从数组中移除第一个或最后一个值。假设Tom的文档中有一个名为siblings的属性，其值为['Marie', 'Bob', 'Kevin', 'Alex']。

```
db.people.update({name: 'Tom'}, {$pop: {siblings: -1}})
// 这将从siblings数组中移除第一个值，在本例中是'Marie'。


db.people.update({name: 'Tom'}, {$pop: {siblings: 1}})
// 这将从 siblings 数组中移除最后一个值，在本例中是 'Alex'。
```

第2.7节：“multi”参数在更新多个文档时的使用

要更新集合中的多个文档，请将 multi 选项设置为 true。

```
db.collection.update(
   query,
update,
   {
upsert: boolean,
     multi: boolean,
     writeConcern: document
   }
)
```

multi 是可选的。如果设置为 true，则更新符合查询条件的多个文档。如果设置为 false，则只更新一个文档。默认值为 false。

> db.mycol.find() { "_id" : ObjectId(598354878df45ec5), "title":"MongoDB 概述"} { "_id" : ObjectId(59835487adf45ec6), "title":"NoSQL 概述"} { "_id" : ObjectId(59835487adf45ec7), "title":"教程点概述"}

> db.mycol.update({'title':'MongoDB 概述'}, {$set:{'title':'新的 MongoDB 教程'}},{multi:true})


```
)
```

Section 2.6: More update operators

You can use other operators besides `$set` when updating a document. The `$push` operator allows you to push a value into an array, in this case we will add a new nickname to the `nicknames` array.

```
db.people.update({name: 'Tom'}, {$push: {nicknames: 'Tommy'}})
// This adds the string 'Tommy' into the nicknames array in Tom's document.
```

The `$pull` operator is the opposite of `$push`, you can pull specific items from arrays.

```
db.people.update({name: 'Tom'}, {$pull: {nicknames: 'Tommy'}})
// This removes the string 'Tommy' from the nicknames array in Tom's document.
```

The `$pop` operator allows you to remove the first or the last value from an array. Let's say Tom's document has a property called siblings that has the value `['Marie', 'Bob', 'Kevin', 'Alex']`.

```
db.people.update({name: 'Tom'}, {$pop: {siblings: -1}})
// This will remove the first value from the siblings array, which is 'Marie' in this case.


db.people.update({name: 'Tom'}, {$pop: {siblings: 1}})
// This will remove the last value from the siblings array, which is 'Alex' in this case.
```

Section 2.7: "multi" Parameter while updating multiple documents

To update multiple documents in a collection, set the multi option to true.

```
db.collection.update(
   query,
   update,
   {
     upsert: boolean,
     multi: boolean,
     writeConcern: document
   }
)
```

multi is optional. If set to true, updates multiple documents that meet the query criteria. If set to false, updates one document. The default value is false.

> db.mycol.find() { "_id" : ObjectId(598354878df45ec5), "title":"MongoDB Overview"} { "_id" : ObjectId(59835487adf45ec6), "title":"NoSQL Overview"} { "_id" : ObjectId(59835487adf45ec7), "title":"Tutorials Point Overview"}

> db.mycol.update({'title':'MongoDB Overview'}, {$set:{'title':'New MongoDB Tutorial'}},{multi:true})

第三章：获取数据库信息

第3.1节：列出数据库中的所有集合

```
show collections
```

或者

```
show tables
```

或者

```
db.getCollectionNames()
```

第3.2节：列出所有数据库

```
show dbs
```

或者

```
db.adminCommand('listDatabases')
```

或者

```
db.getMongo().getDBNames()
```


Chapter 3: Getting database information

Section 3.1: List all collections in database

```
show collections
```

or

```
show tables
```

or

```
db.getCollectionNames()
```

Section 3.2: List all databases

```
show dbs
```

or

```
db.adminCommand('listDatabases')
```

or

```
db.getMongo().getDBNames()
```

第4章：数据查询（入门）

基本查询示例

第4.1节：Find()

检索集合中的所有文档

```
db.collection.find({});
```

使用条件检索集合中的文档（类似于MYSQL中的WHERE）

```
db.collection.find({key: value});
示例
db.users.find({email:"sample@email.com"});
```

使用布尔条件（查询操作符）检索集合中的文档

```
//与 (AND)
db.collection.find( {
    $and: [
     { key: value }, { key: value }
    ]
})
//或
db.collection.find( {
    $or: [
     { key: value }, { key: value }
    ]
})
//非
db.inventory.find( { key: { $not: value } } )
```

更多布尔操作和示例可以在这里找到

注意：find() 即使找到匹配的文档也会继续搜索集合，因此在大型集合中效率较低，然而通过合理建模数据和/或使用索引，可以提高find()的效率

第4.2节：FindOne()

```
db.collection.findOne({});
```

查询功能类似于find()，但一旦找到符合条件的文档就会结束执行，如果使用空对象，则会获取第一个文档并返回。findOne() mongodb api文档

第4.3节：limit、skip、sort和count对find()方法结果的操作

类似于聚合方法，通过 find() 方法你也可以限制、跳过、排序和计数结果。
假设我们有以下集合：

Chapter 4: Querying for Data (Getting Started)

Basic querying examples

Section 4.1: Find()

retrieve all documents in a collection

```
db.collection.find({});
```

retrieve documents in a collection using a condition (similar to WHERE in MYSQL)

```
db.collection.find({key: value});
example
  db.users.find({email:"sample@email.com"});
```

retrieve documents in a collection using Boolean conditions (Query Operators)

```
//AND
db.collection.find( {
    $and: [
     { key: value }, { key: value }
    ]
})
//OR
db.collection.find( {
    $or: [
     { key: value }, { key: value }
    ]
})
//NOT
db.inventory.find( { key: { $not: value } } )
```

more boolean operations and examples can be found here

NOTE: *find()* will keep on searching the collection even if a document match has been found , therefore it is inefficient when used in a large collection , however by carefully modeling your data and/or using indexes you can increase the efficiency of *find()*

Section 4.2: FindOne()

```
db.collection.findOne({});
```

the querying functionality is similar to find() but this will end execution the moment it finds one document matching its condition , if used with and empty object , it will fetch the first document and return it . findOne() mongodb api documentation

Section 4.3: limit, skip, sort and count the results of the find() method

Similar to aggregation methods also by the find() method you have the possibility to limit, skip, sort and count the results. Let say we have following collection:

```
db.test.insertMany([
    {name:"Any", age:"21", status:"busy"},
    {name:"Tony", age:"25", status:"busy"},
    {name:"Bobby", age:"28", status:"online"},
    {name:"Sonny", age:"28", status:"away"},
    {name:"Cher", age:"20", status:"online"}
])
```

列出集合内容：

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

将返回：

```
{ "_id" : ObjectId("592516d7fbd5b591f53237b0"), "name" : "Any", "age" : "21", "status" : "busy" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b1"), "name" : "Tony", "age" : "25", "status" : "busy" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b2"), "name" : "Bobby", "age" : "28", "status" : "online"
}
{ "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : "Sonny", "age" : "28", "status" : "away" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : "Cher", "age" : "20", "status" : "online"
}
```

跳过前三个文档：

```
db.test.find({}).skip(3)
```

将返回：

```
{ "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : "Sonny", "age" : "28", "status" : "away" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : "Cher", "age" : "20", "status" : "online"
}
```

按字段名称降序排序：

```
db.test.find({}).sort({ "name" : -1})
```

将返回：

```
{ "_id" : ObjectId("592516d7fbd5b591f53237b1"), "name" : "Tony", "age" : "25", "status" : "busy" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : "Sonny", "age" : "28", "status" : "away" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : "Cher", "age" : "20", "status" : "online"
}
{ "_id" : ObjectId("592516d7fbd5b591f53237b2"), "name" : "Bobby", "age" : "28", "status" : "online"
}
        { "_id" : ObjectId("592516d7fbd5b591f53237b0"), "name" : "Any", "age" : "21", "status" : "busy" }
```

如果想要升序排序，只需将 -1 替换为 1

统计结果数量：

```
db.test.find({}).count()
```

将返回：

```
5
```



```
db.test.insertMany([
    {name:"Any", age:"21", status:"busy"},
    {name:"Tony", age:"25", status:"busy"},
    {name:"Bobby", age:"28", status:"online"},
    {name:"Sonny", age:"28", status:"away"},
    {name:"Cher", age:"20", status:"online"}
])
```

To list the collection:

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

Will return:

```
{ "_id" : ObjectId("592516d7fbd5b591f53237b0"), "name" : "Any", "age" : "21", "status" : "busy" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b1"), "name" : "Tony", "age" : "25", "status" : "busy" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b2"), "name" : "Bobby", "age" : "28", "status" : "online"
}
{ "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : "Sonny", "age" : "28", "status" : "away" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : "Cher", "age" : "20", "status" : "online"
}
```

To skip first 3 documents:

```
db.test.find({}).skip(3)
```

Will return:

```
{ "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : "Sonny", "age" : "28", "status" : "away" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : "Cher", "age" : "20", "status" : "online"
}
```

To sort descending by the field name:

```
db.test.find({}).sort({ "name" : -1})
```

Will return:

```
{ "_id" : ObjectId("592516d7fbd5b591f53237b1"), "name" : "Tony", "age" : "25", "status" : "busy" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : "Sonny", "age" : "28", "status" : "away" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : "Cher", "age" : "20", "status" : "online"
}
{ "_id" : ObjectId("592516d7fbd5b591f53237b2"), "name" : "Bobby", "age" : "28", "status" : "online"
}
{ "_id" : ObjectId("592516d7fbd5b591f53237b0"), "name" : "Any", "age" : "21", "status" : "busy" }
```

If you want to sort ascending just replace -1 with 1

To count the results:

```
db.test.find({}).count()
```

Will return:

```
5
```

这些方法也可以组合使用。例如，从降序排序的集合中获取 2 个文档
跳过第一个：

```
db.test.find({}).sort({ "name" : -1}).skip(1).limit(2)
```

将返回：

```
{ "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : "Sonny", "age" : "28", "status" : "away" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : "Cher", "age" : "20", "status" : "online"
}
```

第4.4节：查询文档 - 使用 AND、OR 和 IN 条件

来自学生集合的所有文档。

```
> db.students.find().pretty();

{
    "_id" : ObjectId("58f29a694117d1b7af126dca"),
    "studentNo" : 1,
    "firstName" : "Prosen",
    "lastName" : "Ghosh",
    "age" : 25
}

    "_id" : ObjectId("58f29a694117d1b7af126dcb"),
    "studentNo" : 2,
    "firstName" : "Rajib",
    "lastName" : "Ghosh",
    "age" : 25
}

    "_id" : ObjectId("58f29a694117d1b7af126dcc"),
    "studentNo" : 3,
    "firstName" : "里兹维",
    "lastName" : "Amin",
    "age" : 23
}

    "_id" : ObjectId("58f29a694117d1b7af126dcd"),
    "studentNo" : 4,
    "firstName" : "Jabed",
    "lastName" : "Bangali",
    "age" : 25
}

    "_id" : ObjectId("58f29a694117d1b7af126dce"),
    "studentNo" : 5,
    "firstName" : "Gm",
    "lastName" : "Anik",
    "age" : 23
}
```

> 上述命令的类似mySql查询。
>
> ```
> SELECT * FROM students;
> ```


Also combinations of this methods are allowed. For example get 2 documents from descending sorted collection skipping the first 1:

```
db.test.find({}).sort({ "name" : -1}).skip(1).limit(2)
```

Will return:

```
{ "_id" : ObjectId("592516d7fbd5b591f53237b3"), "name" : "Sonny", "age" : "28", "status" : "away" }
{ "_id" : ObjectId("592516d7fbd5b591f53237b4"), "name" : "Cher", "age" : "20", "status" : "online"
}
```

Section 4.4: Query Document - Using AND, OR and IN Conditions

All documents from `students` collection.

```
> db.students.find().pretty();

{
    "_id" : ObjectId("58f29a694117d1b7af126dca"),
    "studentNo" : 1,
    "firstName" : "Prosen",
    "lastName" : "Ghosh",
    "age" : 25
}
{
    "_id" : ObjectId("58f29a694117d1b7af126dcb"),
    "studentNo" : 2,
    "firstName" : "Rajib",
    "lastName" : "Ghosh",
    "age" : 25
}
{
    "_id" : ObjectId("58f29a694117d1b7af126dcc"),
    "studentNo" : 3,
    "firstName" : "Rizve",
    "lastName" : "Amin",
    "age" : 23
}
{
    "_id" : ObjectId("58f29a694117d1b7af126dcd"),
    "studentNo" : 4,
    "firstName" : "Jabed",
    "lastName" : "Bangali",
    "age" : 25
}
{
    "_id" : ObjectId("58f29a694117d1b7af126dce"),
    "studentNo" : 5,
    "firstName" : "Gm",
    "lastName" : "Anik",
    "age" : 23
}
```

> Similar `mySql` Query of the above command.
>
> ```
> SELECT * FROM students;
> ```

```
db.students.find({firstName:"Prosen"});

{ "_id" : ObjectId("58f2547804951ad51ad206f5"), "studentNo" : "1", "firstName" : "Prosen",
"lastName" : "Ghosh", "age" : "23" }
```

> 上述命令的类似mySql查询。
>
> ```
> SELECT * FROM students WHERE firstName = "Prosen";
> ```

AND 查询

```
db.students.find({
    "firstName": "Prosen",
    "age": {
        "$gte": 23
    }
});

{ "_id" : ObjectId("58f29a694117d1b7af126dca"), "studentNo" : 1, "firstName" : "Prosen", "lastName"
: "Ghosh", "age" : 25 }
```

> 上述命令的类似mySql查询。
>
> ```
> SELECT * FROM students WHERE firstName = "Prosen" AND age >= 23
> ```

或查询

```
db.students.find({
     "$or": [{
          "firstName": "Prosen"
     }, {
          "age": {
               "$gte": 23
          }
     }]
 });

{ "_id" : ObjectId("58f29a694117d1b7af126dca"), "studentNo" : 1, "firstName" : "Prosen", "lastName"
: "Ghosh", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcb"), "studentNo" : 2, "firstName" : "Rajib", "lastName"
: "高什", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcc"), "studentNo" : 3, "firstName" : "里兹维", "lastName"
: "阿明", "age" : 23 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcd"), "studentNo" : 4, "firstName" : "贾贝德", "lastName"
: "孟加拉人", "age" : 25 }
        { "_id" : ObjectId("58f29a694117d1b7af126dce"), "studentNo" : 5, "firstName" : "Gm", "lastName" :
"Anik", "age" : 23 }
```

> 上述命令的类似mySql查询。
>
> ```
> SELECT * FROM students WHERE firstName = "Prosen" OR age >= 23
> ```

与 或 查询

```
db.students.find({
```



```
db.students.find({firstName:"Prosen"});

{ "_id" : ObjectId("58f2547804951ad51ad206f5"), "studentNo" : "1", "firstName" : "Prosen",
"lastName" : "Ghosh", "age" : "23" }
```

> Similar `mySql` Query of the above command.
>
> ```
> SELECT * FROM students WHERE firstName = "Prosen";
> ```

AND Queries

```
db.students.find({
    "firstName": "Prosen",
    "age": {
        "$gte": 23
    }
});

{ "_id" : ObjectId("58f29a694117d1b7af126dca"), "studentNo" : 1, "firstName" : "Prosen", "lastName"
: "Ghosh", "age" : 25 }
```

> Similar `mySql` Query of the above command.
>
> ```
> SELECT * FROM students WHERE firstName = "Prosen" AND age >= 23
> ```

Or Queries

```
db.students.find({
     "$or": [{
          "firstName": "Prosen"
     }, {
          "age": {
               "$gte": 23
          }
     }]
 });

{ "_id" : ObjectId("58f29a694117d1b7af126dca"), "studentNo" : 1, "firstName" : "Prosen", "lastName"
: "Ghosh", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcb"), "studentNo" : 2, "firstName" : "Rajib", "lastName"
: "Ghosh", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcc"), "studentNo" : 3, "firstName" : "Rizve", "lastName"
: "Amin", "age" : 23 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcd"), "studentNo" : 4, "firstName" : "Jabed", "lastName"
: "Bangali", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dce"), "studentNo" : 5, "firstName" : "Gm", "lastName" :
"Anik", "age" : 23 }
```

> Similar `mySql` Query of the above command.
>
> ```
> SELECT * FROM students WHERE firstName = "Prosen" OR age >= 23
> ```

And OR Queries

```
db.students.find({
```

```
firstName : "Prosen",
        $or : [
            {age : 23},
            {age : 25}
        ]
});

{ "_id" : ObjectId("58f29a694117d1b7af126dca"), "studentNo" : 1, "firstName" : "Prosen", "lastName"
: "Ghosh", "age" : 25 }
```

> 上述命令的类似 MySQL 查询。
>
> ```
> SELECT * FROM students WHERE firstName = "Prosen" AND age = 23 OR age = 25;
> ```

IN 查询 该查询可以替代多次使用 OR 查询

```
db.students.find(lastName:{$in:["Ghosh", "Amin"]})

{ "_id" : ObjectId("58f29a694117d1b7af126dca"), "studentNo" : 1, "firstName" : "Prosen", "lastName"
: "Ghosh", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcb"), "studentNo" : 2, "firstName" : "Rajib", "lastName"
: "高什", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcc"), "studentNo" : 3, "firstName" : "里兹维", "lastName"
: "阿明", "age" : 23 }
```

> 上述命令的类似 MySQL 查询
>
> ```
> SELECT * FROM students WHERE lastName IN ('Ghosh', 'Amin')
> ```

第4.5节：带投影的find()方法

带投影的find()方法的基本语法如下

```
> db.集合名称.find({},{键:1});
```

如果想显示所有文档但不显示age字段，命令如下

```
db.people.find({},{age : 0});
```

如果想显示所有文档的age字段，命令如下

第4.6节：带投影的Find()方法

在MongoDB中，投影意味着只选择必要的数据，而不是选择文档的全部数据。

带投影的find()方法的基本语法如下

```
> db.集合名称.find({},{键:1});
```

如果想显示所有文档但不显示age字段，命令如下

```
> db.people.find({},{age:0});
```



```
        firstName : "Prosen",
        $or : [
            {age : 23},
            {age : 25}
        ]
});

{ "_id" : ObjectId("58f29a694117d1b7af126dca"), "studentNo" : 1, "firstName" : "Prosen", "lastName"
: "Ghosh", "age" : 25 }
```

> Similar mySql Query of the above command.
>
> ```
> SELECT * FROM students WHERE firstName = "Prosen" AND age = 23 OR age = 25;
> ```

IN Queries This queries can improve multiple use of OR Queries

```
db.students.find(lastName:{$in:["Ghosh", "Amin"]})

{ "_id" : ObjectId("58f29a694117d1b7af126dca"), "studentNo" : 1, "firstName" : "Prosen", "lastName"
: "Ghosh", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcb"), "studentNo" : 2, "firstName" : "Rajib", "lastName"
: "Ghosh", "age" : 25 }
{ "_id" : ObjectId("58f29a694117d1b7af126dcc"), "studentNo" : 3, "firstName" : "Rizve", "lastName"
: "Amin", "age" : 23 }
```

> Similar mySql query to above command
>
> ```
> SELECT * FROM students WHERE lastName IN ('Ghosh', 'Amin')
> ```

Section 4.5: find() method with Projection

The basic syntax of `find()` method with projection is as follows

```
> db.COLLECTION_NAME.find({},{KEY:1});
```

If you want to show all documents without the age field then the command is as follows

```
db.people.find({},{age : 0});
```

If you want to show all documents the age field then the command is as follows

Section 4.6: Find() method with Projection

In MongoDB, projection means selecting only the necessary data rather than selecting whole of the data of a document.

The basic syntax of `find()` method with projection is as follows

```
> db.COLLECTION_NAME.find({},{KEY:1});
```

If you want to to show all document without the age field then the command is as follows

```
> db.people.find({},{age:0});
```

如果你只想显示年龄字段，命令如下

```
> db.people.find({},{age:1});
```

注意：执行find()方法时，_id字段总是会显示，如果你不想显示该字段，则需要将其设置为0。

```
> db.people.find({},{name:1,_id:0});
```

注意：1表示显示字段，0表示隐藏字段。

If you want to show only the age field then the command is as follows

```
> db.people.find({},{age:1});
```

Note: `_id` field is always displayed while executing `find()` method, if you don't want this field, then you need to set it as `0`.

```
> db.people.find({},{name:1,_id:0});
```

Note: `1` is used to show the field while `0` is used to hide the fields.

第5章：更新操作符

参数	含义
字段名	字段将被更新：{name: 'Tom'}
targetVaule值将被分配给字段：{name: 'Tom'}	

第5.1节：使用 $set 操作符更新文档中的指定字段

一、概述

MongoDB 与关系型数据库管理系统（RDBMS）之间的一个显著区别是，MongoDB 有多种操作符。其中之一是更新操作符，用于更新语句中。

二、如果不使用更新操作符会发生什么？

假设我们有一个student集合来存储学生信息（表格视图）：

age	name	sex
20	Tom	M
25	Billy	M
18	Mary	F
40	Ken	M

有一天你接到一个任务，需要将 Tom 的性别从“M”改为“F”。这很简单，对吧？所以你根据 RDBMS 的经验很快写了下面的语句：

```
db.student.update(
    {name: 'Tom'}, // 查询条件
    {sex: 'F'} // 更新操作
);
```

让我们看看结果是什么：

age	name	sex
		F
25	Billy	M
18	Mary	F
40	Ken	M

我们丢失了 Tom 的年龄和名字！通过这个例子，我们可以知道如果更新语句中没有任何更新操作符，整个文档将被覆盖。这是 MongoDB 的默认行为。

Chapter 5: Update Operators

parameters	Meaning
fieldName	Field will be updated :{***name***: 'Tom'}
targetVaule	Value will be assigned to the field :{name: ***'Tom'***}

Section 5.1: $set operator to update specified field(s) in document(s)

I.Overview

A significant difference between MongoDB & RDBMS is MongoDB has many kinds of operators. One of them is update operator, which is used in update statements.

II.What happen if we don't use update operators?

Suppose we have a **student** collection to store student information(Table view):

age	name	sex
20	Tom	M
25	Billy	M
18	Mary	F
40	Ken	M

One day you get a job that need to change Tom's gender from "M" to "F". That's easy, right? So you write below statement very quickly based on your RDBMS experience:

```
db.student.update(
    {name: 'Tom'}, // query criteria
    {sex: 'F'} // update action
);
```

Let's see what is the result:

age	name	sex
		F
25	Billy	M
18	Mary	F
40	Ken	M

We lost Tom's age & name! From this example, we can know that **the whole document will be overrided** if without any update operator in update statement. This is the default behavior of MongoDB.

III.$set 操作符

如果我们只想修改 Tom 文档中的“sex”字段，可以使用$set来指定要更新的字段：

```
db.student.update(
    {name: 'Tom'}, // 查询条件
    {$set: {sex: 'F'}} // 更新操作
);
```

$set的值是一个对象，其字段表示你想要更新的文档中的字段，这些字段的值是目标值。

所以，现在结果是正确的：

age	name	sex
20	Tom	F
25	Billy	M
18	Mary	F
40	Ken	M

另外，如果你想同时修改“sex”和“age”，可以将它们一起添加到$set中：

```
db.student.update(
    {name: 'Tom'}, // 查询条件
    {$set: {sex: 'F', age: 40}} // 更新操作
);
```


III.$set operator

If we want to change only the 'sex' field in Tom's document, we can use `$set` to specify which field(s) we want to update:

```
db.student.update(
    {name: 'Tom'}, // query criteria
    {$set: {sex: 'F'}} // update action
);
```

The value of `$set` is an object, its fields stands for those fields you want to update in the documents, and the values of these fields are the target values.

So, the result is correct now:

age	name	sex
20	Tom	F
25	Billy	M
18	Mary	F
40	Ken	M

Also, if you want to change both 'sex' and 'age' at the same time, you can append them to `$set` :

```
db.student.update(
    {name: 'Tom'}, // query criteria
    {$set: {sex: 'F', age: 40}} // update action
);
```

第6章：Upserts 和 Inserts

第6.1节：插入文档

_id 是一个12字节的十六进制数字，用于确保每个文档的唯一性。您可以在插入文档时提供 _id。如果您没有提供，MongoDB 会为每个文档提供一个唯一的id。这12字节中，前4字节表示当前时间戳，接下来的3字节表示机器id，接下来的2字节表示mongodb服务器的进程id，
剩余的3字节是简单的递增值。

```
db.mycol.insert({
_id: ObjectId(7df78ad8902c),
 title: 'MongoDB 概述',
 description: 'MongoDB 是一个 NoSQL 数据库',
 by: 'tutorials point',
url: 'http://www.tutorialspoint.com',
 tags: ['mongodb', '数据库', 'NoSQL'],
 likes: 100
})
```

这里 mycol 是集合名称，如果数据库中不存在该集合，MongoDB 会创建该集合，然后将文档插入其中。在插入的文档中，如果我们没有指定 _id 参数，MongoDB 会为该文档分配一个唯一的 ObjectId。

Chapter 6: Upserts and Inserts

Section 6.1: Insert a document

_id is a 12 bytes hexadecimal number which assures the uniqueness of every document. You can provide _id while inserting the document. **If you didn't provide then MongoDB provide a unique id for every document.** These 12 bytes first 4 bytes for the current timestamp, next 3 bytes for machine id, next 2 bytes for process id of mongodb server and remaining 3 bytes are simple incremental value.

```
db.mycol.insert({
 _id: ObjectId(7df78ad8902c),
 title: 'MongoDB Overview',
 description: 'MongoDB is no sql database',
 by: 'tutorials point',
 url: 'http://www.tutorialspoint.com',
 tags: ['mongodb', 'database', 'NoSQL'],
 likes: 100
})
```

Here *mycol* is a collection name, if the collection doesn't exist in the database, then MongoDB will create this collection and then insert document into it. In the inserted document if we don't specify the *_id* parameter, then MongoDB assigns an unique ObjectId for this document.

第7章：集合

第7.1节：创建集合

首先选择或创建一个数据库。

```
> use mydb
切换到数据库 mydb
```

使用 db. createCollection("yourCollectionName") 方法可以显式创建集合。

```
> db. createCollection("newCollection1")
{ "ok" : 1 }
```

使用 show collections 命令查看数据库中的所有集合。

```
> show collections
newCollection1
system.indexes
>
```

db. createCollection() 方法具有以下参数：

参数	类型	描述
名称	字符串	要创建的集合名称。
选项	document	可选。用于创建有上限的集合或预分配新集合空间的配置选项。

下面的示例展示了createCollection()方法的语法及一些重要选项

```
>db.createCollection("newCollection4", {capped :true, autoIndexId : true, size : 6142800, max :
10000})
{ "ok" : 1 }
```

无论是 db.collection.insert()还是 db.collection.createIndex()操作，如果对应集合不存在，都会自动创建该集合。

```
> db.newCollection2.insert({name : "XXX"})
> db.newCollection3.createIndex({accountNo : 1})
```

现在，使用show collections命令显示所有集合

```
> show collections
newCollection1
newCollection2
newCollection3
newCollection4
system.indexes
```

如果想查看插入的文档，使用find()命令。

```
> db.newCollection2.find()
{ "_id" : ObjectId("58f26876cabafaeb509e9c1f"), "name" : "XXX" }
```


Chapter 7: Collections

Section 7.1: Create a Collection

First Select Or Create a database.

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

Using `db.createCollection("yourCollectionName")` method you can explicitly create Collection.

```
> db.createCollection("newCollection1")
{ "ok" : 1 }
```

Using `show collections` command see all collections in the database.

```
> show collections
newCollection1
system.indexes
>
```

The `db.createCollection()` method has the following parameters:

Parameter	Type	Description
name	string	The name of the collection to create.
options	document	*Optional.* Configuration options for creating a capped collection or for preallocating space in a new collection.

The fllowing example shows the syntax of `createCollection()` method with few important options

```
>db.createCollection("newCollection4", {capped :true, autoIndexId : true, size : 6142800, max :
10000})
{ "ok" : 1 }
```

Both the `db.collection.insert()` and the `db.collection.createIndex()` operations create their respective collection if they do not already exist.

```
> db.newCollection2.insert({name : "XXX"})
> db.newCollection3.createIndex({accountNo : 1})
```

Now, Show All the collections using `show collections` command

```
> show collections
newCollection1
newCollection2
newCollection3
newCollection4
system.indexes
```

If you want to see the inserted document, use the `find()` command.

```
> db.newCollection2.find()
{ "_id" : ObjectId("58f26876cabafaeb509e9c1f"), "name" : "XXX" }
```

第7.2节：删除集合

MongoDB的 db.collection.drop() 用于从数据库中删除集合。

首先，检查数据库 mydb 中可用的集合。

```
> use mydb
切换到数据库 mydb

> show collections
newCollection1
newCollection2
newCollection3
system.indexes
```

现在删除名为 newCollection1 的集合。

```
> db.newCollection1.drop()
true
```

注意： 如果集合删除成功，则该方法返回 true，否则返回 false。

再次检查数据库中的集合列表。

```
> 显示集合
newCollection2
newCollection3
system.索引
```

参考： MongoDB drop() 方法。

Section 7.2: Drop Collection

MongoDB's `db.collection.drop()` is used to drop a collection from the database.

First, check the available collections into your database `mydb`.

```
> use mydb
switched to db mydb

> show collections
newCollection1
newCollection2
newCollection3
system.indexes
```

Now drop the collection with the name `newCollection1`.

```
> db.newCollection1.drop()
true
```

Note: If the collection droped successfully then the method will return **`true`** otherwise it will return **`false`**.

Again check the list of collections into database.

```
> show collections
newCollection2
newCollection3
system.indexes
```

Reference: MongoDB drop() Method.

第8章：聚合

参数	详情
管道	数组（一系列数据聚合操作或阶段）
选项	文档（可选，仅当管道以数组形式存在时可用）

聚合操作处理数据记录并返回计算结果。聚合操作将来自多个文档的值分组在一起，并可以对分组数据执行各种操作以返回单个结果。MongoDB 提供三种执行聚合的方法：聚合管道、Map-Reduce 函数和单一目的聚合方法。

摘自 Mongo 手册 https://docs.mongodb.com/manual/aggregation/

第8.1节：计数

如何获取借方和贷方交易的数量？一种方法是使用count()函数，如下所示。

```
> db.transactions.count({cr_dr : "D"});
```

或者

```
> db.transactions.find({cr_dr : "D"}).length();
```

但是如果你事先不知道cr_dr的可能取值怎么办？这时聚合框架就派上用场了。请看下面的聚合查询。

```
> db.transactions.aggregate(
      [
          {
$group : {
_id : '$cr_dr', // 按交易类型分组
                 // 对每个文档加1，统计该类型交易的数量
                count : {$sum : 1}
              }
          }
      ]
  );
```

结果如下

```
{
    "_id" : "C",
    "count" : 3
}
{
    "_id" : "D",
    "count" : 5
}
```

第8.2节：求和

如何获取amount的总和？请参见下面的聚合查询。

Chapter 8: Aggregation

Parameter	Details
pipeline	array(A sequence of data aggregation operations or stages)
options	document(optional, available only if pipeline present as an array)

`Aggregations` operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. MongoDB provides three ways to perform aggregation: the aggregation pipeline, the map-reduce function, and single purpose aggregation methods.

From Mongo manual https://docs.mongodb.com/manual/aggregation/

Section 8.1: Count

How do you get the number of Debit and Credit transactions? One way to do it is by using `count()` function as below.

```
> db.transactions.count({cr_dr : "D"});
```

or

```
> db.transactions.find({cr_dr : "D"}).length();
```

But what if you do not know the possible values of `cr_dr` upfront. Here Aggregation framework comes to play. See the below Aggregate query.

```
> db.transactions.aggregate(
      [
          {
              $group : {
                  _id : '$cr_dr', // group by type of transaction
                // Add 1 for each document to the count for this type of transaction
                  count : {$sum : 1}
              }
          }
      ]
  );
```

And the result is

```
{
    "_id" : "C",
    "count" : 3
}
{
    "_id" : "D",
    "count" : 5
}
```

Section 8.2: Sum

How to get the summation of `amount`? See the below aggregate query.

```
> db.transactions.aggregate(
    [
        {
$group : {
_id : '$cr_dr',
count : {$sum : 1},    // 计数
                totalAmount : {$sum : '$amount'}    // 求金额总和
            }
        }
    ]
 );
```

结果如下

```
{
    "_id" : "C",
    "count" : 3.0,
    "totalAmount" : 120.0
}
{
    "_id" : "D",
    "count" : 5.0,
    "totalAmount" : 410.0
}
```

另一种版本，汇总金额和费用。

```
> db.transactions.aggregate(
     [
         {
$group : {
_id : '$cr_dr',
计数 : {$sum : 1},
                总金额 : {$sum : { $sum : ['$amount', '$fee']}}
            }
        }
     ]
  );
```

结果如下

```
{
    "_id" : "C",
    "计数" : 3.0,
    "总金额" : 128.0
}
{
    "_id" : "D",
    "计数" : 5.0,
    "总金额" : 422.0
}
```

第8.3节：平均值

如何获得借记和贷记交易的平均金额？

```
> db.transactions.aggregate(
    [
        {
```



```
> db.transactions.aggregate(
    [
        {
            $group : {
                _id : '$cr_dr',
                count : {$sum : 1},    //counts the number
                totalAmount : {$sum : '$amount'}    //sums the amount
            }
        }
    ]
 );
```

And the result is

```
{
    "_id" : "C",
    "count" : 3.0,
    "totalAmount" : 120.0
}
{
    "_id" : "D",
    "count" : 5.0,
    "totalAmount" : 410.0
}
```

Another version that sums `amount` and `fee`.

```
> db.transactions.aggregate(
     [
         {
             $group : {
                 _id : '$cr_dr',
                 count : {$sum : 1},
                 totalAmount : {$sum : { $sum : ['$amount', '$fee']}}
             }
         }
     ]
  );
```

And the result is

```
{
    "_id" : "C",
    "count" : 3.0,
    "totalAmount" : 128.0
}
{
    "_id" : "D",
    "count" : 5.0,
    "totalAmount" : 422.0
}
```

Section 8.3: Average

How to get the average amount of debit and credit transactions?

```
> db.transactions.aggregate(
    [
        {
```

```
$group : {
_id : '$cr_dr', // 按交易类型（借记或贷记）分组
              计数 : {$sum : 1},     // 每种类型的交易数量
              总金额 : {$sum : { $sum : ['$amount', '$fee']}},     // 总和
              平均金额 : {$avg : { $sum : ['$amount', '$fee']}}    // 平均值
            }
        }
    ]
)
```

结果是

```
{
    "_id" : "C", // 信用交易金额
    "count" : 3.0,
    "totalAmount" : 128.0,
    "averageAmount" : 40.0
}
{
    "_id" : "D", // 借记交易的金额
    "count" : 5.0,
    "totalAmount" : 422.0,
    "averageAmount" : 82.0
}
```

第8.4节：数组操作

当你想操作数组中的数据条目时，首先需要对数组进行unwind操作。unwind操作会为数组中的每个条目创建一个文档。当你有大量包含大数组的文档时，你会看到文档数量的爆炸性增长。

```
{ "_id" : 1, "item" : "myItem1", sizes: [ "S", "M", "L"] }
{ "_id" : 2, "item" : "myItem2", sizes: [ "XS", "M", "XL"] }

db.inventory.aggregate( [ { $unwind : "$sizes" }] )
```

一个重要的注意事项是，当文档不包含该数组时，该文档将会丢失。从Mongo 3.2版本开始，增加了一个unwind选项“preserveNullAndEmptyArrays”。该选项确保当数组缺失时，文档得以保留。

```
{ "_id" : 1, "item" : "myItem1", sizes: [ "S", "M", "L"] }
{ "_id" : 2, "item" : "myItem2", sizes: [ "XS", "M", "XL"] }
{ "_id" : 3, "item" : "myItem3" }

db.inventory.aggregate( [ { $unwind : { path: "$sizes", includeArrayIndex: "arrayIndex" } }] )
```

第8.5节：适用于工作和学习的聚合查询示例

聚合用于执行mongo查询中无法通过普通“find”查询完成的复杂数据搜索操作。

创建一些示例数据：

```
db.employees.insert({"name":"Adma","dept":"Admin","languages":["german","french","english","hindi"]
,"age":30, "totalExp":10});
db.employees.insert({"name":"Anna","dept":"Admin","languages":["english","hindi"],"age":35,
```



```
            $group : {
                _id : '$cr_dr', // group by type of transaction (debit or credit)
                count : {$sum : 1},     // number of transaction for each type
                totalAmount : {$sum : { $sum : ['$amount', '$fee']}},     // sum
                averageAmount : {$avg : { $sum : ['$amount', '$fee']}}    // average
            }
        }
    ]
)
```

The result is

```
{
    "_id" : "C", // Amounts for credit transactions
    "count" : 3.0,
    "totalAmount" : 128.0,
    "averageAmount" : 40.0
}
{
    "_id" : "D", // Amounts for debit transactions
    "count" : 5.0,
    "totalAmount" : 422.0,
    "averageAmount" : 82.0
}
```

Section 8.4: Operations with arrays

When you want to work with the data entries in arrays you first need to unwind the array. The unwind operation creates a document for each entry in the array. When you have lot's of documents with large arrays you will see an explosion in number of documents.

```
{ "_id" : 1, "item" : "myItem1", sizes: [ "S", "M", "L"] }
{ "_id" : 2, "item" : "myItem2", sizes: [ "XS", "M", "XL"] }

db.inventory.aggregate( [ { $unwind : "$sizes" }] )
```

An important notice is that when a document doesn't contain the array it will be lost. From mongo 3.2 and up there are is an unwind option "preserveNullAndEmptyArrays" added. This option makes sure the document is preserved when the array is missing.

```
{ "_id" : 1, "item" : "myItem1", sizes: [ "S", "M", "L"] }
{ "_id" : 2, "item" : "myItem2", sizes: [ "XS", "M", "XL"] }
{ "_id" : 3, "item" : "myItem3" }

db.inventory.aggregate( [ { $unwind : { path: "$sizes", includeArrayIndex: "arrayIndex" } }] )
```

Section 8.5: Aggregate query examples useful for work and learning

Aggregation is used to perform complex data search operations in the mongo query which can't be done in normal "find" query.

Create some dummy data:

```
db.employees.insert({"name":"Adma","dept":"Admin","languages":["german","french","english","hindi"]
,"age":30, "totalExp":10});
db.employees.insert({"name":"Anna","dept":"Admin","languages":["english","hindi"],"age":35,
```

```
"totalExp":11});
db.employees.insert({"name":"Bob","dept":"Facilities","languages":["english","hindi"],"age":36,
"totalExp":14});
db.employees.insert({"name":"Cathy","dept":"Facilities","languages":["hindi"],"age":31,
"totalExp":4});
db.employees.insert({"name":"Mike","dept":"HR","languages":["english", "hindi",
"spanish"],"age":26, "totalExp":3});
db.employees.insert({"姓名":"珍妮","部门":"人力资源","语言":["英语", "印地语",
"西班牙语"],"年龄":25, "总经验":3});
```

按主题分类的示例：

1. 匹配（Match）： 用于匹配文档（类似SQL中的where子句）

```
db.employees.aggregate([{$match:{dept:"Admin"}}]);
输出:
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "Admin", "languages" : [
"german", "french", "english", "hindi" ], "age" : 30, "totalExp" : 10 }
{ "_id" : ObjectId("54982fc92e9b4b54ec384a0e"), "name" : "Anna", "dept" : "Admin", "languages" : [
"english", "hindi" ], "age" : 35, "totalExp" : 11 }
```

2. 投影（Project）： 用于填充特定字段的值

投影阶段会自动包含_id字段，除非你指定禁用。

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1}}]);
输出:
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "Admin" }
{ "_id" : ObjectId("54982fc92e9b4b54ec384a0e"), "name" : "Anna", "dept" : "Admin" }

db.employees.aggregate({$project: {'_id':0, 'name': 1}})
输出:
{ "name" : "Adma" }
{ "name" : "Anna" }
{ "name" : "Bob" }
{ "name" : "Cathy" }
{ "name" : "Mike" }
{ "name" : "Jenny" }
```

3. 分组： $group 用于按特定字段对文档进行分组，这里是按“dept”字段的值对文档进行分组。另一个有用的功能是你可以按 null 分组，这意味着所有文档将被聚合到一个组中。

```
db.employees.aggregate([{$group:{"_id":"$dept"}}]);

{ "_id" : "人力资源部" }

{ "_id" : "设施部" }

{ "_id" : "行政部" }

db.employees.aggregate([{$group:{"_id":null, "totalAge":{$sum:"$age"}}}]);
输出:
{ "_id" : null, "noOfEmployee" : 183 }
```

4. 求和： $sum 用于计算组内的值的数量或求和。

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfDept":{$sum:1}}}]);
输出:
```



```
"totalExp":11});
db.employees.insert({"name":"Bob","dept":"Facilities","languages":["english","hindi"],"age":36,
"totalExp":14});
db.employees.insert({"name":"Cathy","dept":"Facilities","languages":["hindi"],"age":31,
"totalExp":4});
db.employees.insert({"name":"Mike","dept":"HR","languages":["english", "hindi",
"spanish"],"age":26, "totalExp":3});
db.employees.insert({"name":"Jenny","dept":"HR","languages":["english", "hindi",
"spanish"],"age":25, "totalExp":3});
```

Examples by topic:

1. Match: Used to match documents (like SQL where clause)

```
db.employees.aggregate([{$match:{dept:"Admin"}}]);
Output:
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "Admin", "languages" : [
"german", "french", "english", "hindi" ], "age" : 30, "totalExp" : 10 }
{ "_id" : ObjectId("54982fc92e9b4b54ec384a0e"), "name" : "Anna", "dept" : "Admin", "languages" : [
"english", "hindi" ], "age" : 35, "totalExp" : 11 }
```

2. Project: Used to populate specific field's value(s)

project stage will include _id field automatically unless you specify to disable.

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1}}]);
Output:
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "Admin" }
{ "_id" : ObjectId("54982fc92e9b4b54ec384a0e"), "name" : "Anna", "dept" : "Admin" }

db.employees.aggregate({$project: {'_id':0, 'name': 1}})
Output:
{ "name" : "Adma" }
{ "name" : "Anna" }
{ "name" : "Bob" }
{ "name" : "Cathy" }
{ "name" : "Mike" }
{ "name" : "Jenny" }
```

3. Group: $group is used to group documents by specific field, here documents are grouped by "dept" field's value. Another useful feature is that you can group by null, it means all documents will be aggregated into one.

```
db.employees.aggregate([{$group:{"_id":"$dept"}}]);

{ "_id" : "HR" }

{ "_id" : "Facilities" }

{ "_id" : "Admin" }

db.employees.aggregate([{$group:{"_id":null, "totalAge":{$sum:"$age"}}}]);
Output:
{ "_id" : null, "noOfEmployee" : 183 }
```

4. Sum: $sum is used to count or sum the values inside a group.

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfDept":{$sum:1}}}]);
Output:
```

```
{ "_id" : "人力资源部", "noOfDept" : 2 }
{ "_id" : "设施部", "noOfDept" : 2 }
{ "_id" : "行政部", "noOfDept" : 2 }
```

5. 平均值： 计算每个组中特定字段值的平均值。

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfEmployee":{$sum:1},
"avgExp":{$avg:"$totalExp"}}}]);
输出:
{ "_id" : "人力资源部", "noOfEmployee" : 2, "totalExp" : 3 }
{ "_id" : "设施部", "noOfEmployee" : 2, "totalExp" : 9 }
{ "_id" : "行政部", "noOfEmployee" : 2, "totalExp" : 10.5 }
```

6. 最小值： 查找每个组中某字段的最小值。

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfEmployee":{$sum:1},
"minExp":{$min:"$totalExp"}}}]);
输出:
{ "_id" : "人力资源部", "noOfEmployee" : 2, "totalExp" : 3 }
{ "_id" : "设施部", "noOfEmployee" : 2, "totalExp" : 4 }
{ "_id" : "行政部", "noOfEmployee" : 2, "totalExp" : 10 }
```

7. 最大值： 查找每个组中某字段的最大值。

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfEmployee":{$sum:1},
"maxExp":{$max:"$totalExp"}}}]);
输出:
{ "_id" : "人力资源部", "noOfEmployee" : 2, "totalExp" : 3 }
{ "_id" : "设施部", "noOfEmployee" : 2, "totalExp" : 14 }
{ "_id" : "行政部", "noOfEmployee" : 2, "totalExp" : 11 }
```

8. 获取每个组中第一个和最后一个文档的特定字段值： 当文档结果已排序时效果良好。

```
db.employees.aggregate([{$group:{"_id":"$age", "lasts":{$last:"$name"},
"firsts":{$first:"$name"}}}]);
输出:
{ "_id" : 25, "lasts" : "珍妮", "firsts" : "珍妮" }
{ "_id" : 26, "lasts" : "迈克", "firsts" : "迈克" }
{ "_id" : 35, "lasts" : "凯茜", "firsts" : "安娜" }
{ "_id" : 30, "lasts" : "亚当", "firsts" : "亚当" }
```

9. 最小值与最大值：

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfEmployee":{$sum:1},
"maxExp":{$max:"$totalExp"}, "minExp":{$min: "$totalExp"}}}]);
输出:
{ "_id" : "人力资源部", "noOfEmployee" : 2, "maxExp" : 3, "minExp" : 3 }
{ "_id" : "设施部", "noOfEmployee" : 2, "maxExp" : 14, "minExp" : 4 }
{ "_id" : "行政部", "noOfEmployee" : 2, "maxExp" : 11, "minExp" : 10 }
```

10. Push 和 addToSet： Push 会将每个文档中某字段的值添加到一个数组中，用于以数组格式投影数据，addToSet 类似于 push，但会去除重复值。

```
db.employees.aggregate([{$group:{"_id":"dept", "arrPush":{$push:"$age"}, "arrSet":
{$addToSet:"$age"}}}]);
输出:
```



```
{ "_id" : "HR", "noOfDept" : 2 }
{ "_id" : "Facilities", "noOfDept" : 2 }
{ "_id" : "Admin", "noOfDept" : 2 }
```

5. Average: Calculates average of specific field's value per group.

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfEmployee":{$sum:1},
"avgExp":{$avg:"$totalExp"}}}]);
Output:
{ "_id" : "HR", "noOfEmployee" : 2, "totalExp" : 3 }
{ "_id" : "Facilities", "noOfEmployee" : 2, "totalExp" : 9 }
{ "_id" : "Admin", "noOfEmployee" : 2, "totalExp" : 10.5 }
```

6. Minimum: Finds minimum value of a field in each group.

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfEmployee":{$sum:1},
"minExp":{$min:"$totalExp"}}}]);
Output:
{ "_id" : "HR", "noOfEmployee" : 2, "totalExp" : 3 }
{ "_id" : "Facilities", "noOfEmployee" : 2, "totalExp" : 4 }
{ "_id" : "Admin", "noOfEmployee" : 2, "totalExp" : 10 }
```

7. Maximum: Finds maximum value of a field in each group.

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfEmployee":{$sum:1},
"maxExp":{$max:"$totalExp"}}}]);
Output:
{ "_id" : "HR", "noOfEmployee" : 2, "totalExp" : 3 }
{ "_id" : "Facilities", "noOfEmployee" : 2, "totalExp" : 14 }
{ "_id" : "Admin", "noOfEmployee" : 2, "totalExp" : 11 }
```

8. Getting specific field's value from first and last document of each group: Works well when doucument result is sorted.

```
db.employees.aggregate([{$group:{"_id":"$age", "lasts":{$last:"$name"},
"firsts":{$first:"$name"}}}]);
Output:
{ "_id" : 25, "lasts" : "Jenny", "firsts" : "Jenny" }
{ "_id" : 26, "lasts" : "Mike", "firsts" : "Mike" }
{ "_id" : 35, "lasts" : "Cathy", "firsts" : "Anna" }
{ "_id" : 30, "lasts" : "Adma", "firsts" : "Adma" }
```

9. Minumum with maximum:

```
db.employees.aggregate([{$group:{"_id":"$dept", "noOfEmployee":{$sum:1},
"maxExp":{$max:"$totalExp"}, "minExp":{$min: "$totalExp"}}}]);
Output:
{ "_id" : "HR", "noOfEmployee" : 2, "maxExp" : 3, "minExp" : 3 }
{ "_id" : "Facilities", "noOfEmployee" : 2, "maxExp" : 14, "minExp" : 4 }
{ "_id" : "Admin", "noOfEmployee" : 2, "maxExp" : 11, "minExp" : 10 }
```

10. Push and addToSet: Push adds a field's value form each document in group to an array used to project data in array format, addToSet is simlar to push but it omits duplicate values.

```
db.employees.aggregate([{$group:{"_id":"dept", "arrPush":{$push:"$age"}, "arrSet":
{$addToSet:"$age"}}}]);
Output:
```

```
{ "_id" : "dept", "arrPush" : [ 30, 35, 35, 35, 26, 25 ], "arrSet" : [ 25, 26, 35, 30 ] }
```

11. Unwind： 用于为指定数组类型字段中的每个值创建多个内存中的文档，然后我们可以基于这些值进行进一步的聚合。

```
db.employees.aggregate([{$match:{"name":"Adma"}}, {$unwind:"$languages"}]);
输出:
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "人力资源部", "languages" :
"德语", "age" : 30, "totalExp" : 10 }
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "人力资源部", "languages" :
"法语", "age" : 30, "totalExp" : 10 }
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "人力资源部", "languages" :
"英语", "age" : 30, "totalExp" : 10 }
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "人力资源部", "languages" :
"印地语", "age" : 30, "totalExp" : 10 }
```

12. 排序：

```
db.employees.aggregate([{$match:{dept:"行政部"}}, {$project:{"name":1, "dept":1}}, {$sort: {name:
1}}]);
输出:
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "行政部" }
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "行政部" }

db.employees.aggregate([{$match:{dept:"行政部"}}, {$project:{"name":1, "dept":1}}, {$sort: {name:
-1}}]);
输出:
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "姓名" : "安娜", "部门" : "管理部" }
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "姓名" : "阿德玛", "部门" : "管理部" }
```

13. 跳过：

```
db.employees.aggregate([{$match:{dept:"行政部"}}, {$project:{"name":1, "dept":1}}, {$sort: {name:
-1}}, {$skip:1}]);
输出:
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "姓名" : "阿德玛", "部门" : "管理部" }
```

14. 限制：

```
db.employees.aggregate([{$match:{dept:"行政部"}}, {$project:{"name":1, "dept":1}}, {$sort: {name:
-1}}, {$limit:1}]);
                                                    输出:

{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "姓名" : "安娜", "部门" : "管理部" }
```

15. 投影中的比较操作符：

```
        db.employees.aggregate([{$match:{部门:"管理部"}}, {$project:{"姓名":1, "部门":1, 年龄: {$gt: ["$年龄",
30]}}}]);
输出:
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "Admin", "age" : false }
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin", "age" : true }
```

16. 匹配中的比较操作符：

```
db.employees.aggregate([{$match:{dept:"Admin", age: {$gt:30}}}, {$project:{"name":1, "dept":1}}]);

输出:
```



```
{ "_id" : "dept", "arrPush" : [ 30, 35, 35, 35, 26, 25 ], "arrSet" : [ 25, 26, 35, 30 ] }
```

11. Unwind: Used to create multiple in-memory documents for each value in the specified array type field, then we can do further aggregation based on those values.

```
db.employees.aggregate([{$match:{"name":"Adma"}}, {$unwind:"$languages"}]);
Output:
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "HR", "languages" :
"german", "age" : 30, "totalExp" : 10 }
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "HR", "languages" :
"french", "age" : 30, "totalExp" : 10 }
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "HR", "languages" :
"english", "age" : 30, "totalExp" : 10 }
{ "_id" : ObjectId("54982fac2e9b4b54ec384a0d"), "name" : "Adma", "dept" : "HR", "languages" :
"hindi", "age" : 30, "totalExp" : 10 }
```

12. Sorting:

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1}}, {$sort: {name:
1}}]);
Output:
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "Admin" }
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin" }

db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1}}, {$sort: {name:
-1}}]);
Output:
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin" }
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "Admin" }
```

13. Skip:

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1}}, {$sort: {name:
-1}}, {$skip:1}]);
Output:
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "Admin" }
```

14. Limit:

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1}}, {$sort: {name:
-1}}, {$limit:1}]);
Output:

{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin" }
```

15. Comparison operator in projection:

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1, age: {$gt: ["$age",
30]}}}]);
Output:
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "Admin", "age" : false }
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin", "age" : true }
```

16. Comparison operator in match:

```
db.employees.aggregate([{$match:{dept:"Admin", age: {$gt:30}}}, {$project:{"name":1, "dept":1}}]);

Output:
```

```
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "姓名" : "安娜", "部门" : "管理部" }
```

> 比较操作符列表：$cmp，$eq，$gt，$gte，$lt，$lte 和 $ne

17. 投影中的布尔聚合操作符：

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1, age: { $and: [ {
$gt: [ "$age", 30 ] }, { $lt: [ "$age", 36 ] } ] }}}]);

输出:
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "Admin", "age" : false }

{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin", "age" : true }
```

18. 匹配中的布尔聚合操作符：

```
db.employees.aggregate([{$match:{dept:"Admin", $and: [{age: { $gt:  30 }}, {age: {$lt: 36 }} ] }},
{$project:{"name":1, "dept":1, age: { $and: [ { $gt: [ "$age", 30 ] }, { $lt: [ "$age", 36 ] } ]
}}}]);
输出:
        { "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin", "age" : true }
```

> 布尔聚合运算符列表：$and、$or 和 $not。

完整参考：https://docs.mongodb.com/v3.2/reference/operator/aggregation/

第8.6节：匹配

如何编写查询以获取所有部门，这些部门中薪资低于或等于70000美元的员工的平均年龄大于或等于35？

为此，我们需要编写一个查询，匹配薪资低于或等于70000美元的员工。然后添加聚合阶段，将员工按部门分组。接着添加一个累加器，字段名例如average_age，使用$avg累加器计算每个部门的平均年龄。在现有的$match和$group聚合之后，再添加一个$match聚合，以便只检索average_age大于或等于35的结果。

```
db.employees.aggregate([
  {"$match": {"salary": {"$lte": 70000}}},
  {"$group": {"_id": "$dept",
              "average_age": {"$avg": "$age"}
             }
  },
  {"$match": {"average_age": {"$gte": 35}}}
])
```

结果是：

```
{
  "_id": "IT",
  "average_age": 31
}
{
  "_id": "客户服务",
```



```
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin" }
```

> List of comparison operators: $cmp, $eq, $gt, $gte, $lt, $lte, and $ne

17. Boolean aggregation opertor in projection:

```
db.employees.aggregate([{$match:{dept:"Admin"}}, {$project:{"name":1, "dept":1, age: { $and: [ {
$gt: [ "$age", 30 ] }, { $lt: [ "$age", 36 ] } ] }}}]);

Output:
{ "_id" : ObjectId("57ff3e553dedf0228d4862ac"), "name" : "Adma", "dept" : "Admin", "age" : false }

{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin", "age" : true }
```

18. Boolean aggregation opertor in match:

```
db.employees.aggregate([{$match:{dept:"Admin", $and: [{age: { $gt:  30 }}, {age: {$lt: 36 }} ] }},
{$project:{"name":1, "dept":1, age: { $and: [ { $gt: [ "$age", 30 ] }, { $lt: [ "$age", 36 ] } ]
}}}]);
Output:
{ "_id" : ObjectId("57ff3e5e3dedf0228d4862ad"), "name" : "Anna", "dept" : "Admin", "age" : true }
```

> List of boolean aggregation opertors: $and, $or, and $not.

Complete refrence: https://docs.mongodb.com/v3.2/reference/operator/aggregation/

Section 8.6: Match

How to write a query to get all departments where average age of employees making less than or $70000 is greather than or equal to 35?

In order to that we need to write a query to match employees that have a salary that is less than or equal to $70000. Then add the aggregate stage to group the employees by the department. Then add an accumulator with a field named e.g. average_age to find the average age per department using the $avg accumulator and below the existing $match and $group aggregates add another $match aggregate so that we're only retrieving results with an average_age that is greather than or equal to 35.

```
db.employees.aggregate([
  {"$match": {"salary": {"$lte": 70000}}},
  {"$group": {"_id": "$dept",
              "average_age": {"$avg": "$age"}
             }
  },
  {"$match": {"average_age": {"$gte": 35}}}
])
```

The result is:

```
{
  "_id": "IT",
  "average_age": 31
}
{
  "_id": "Customer Service",
```

```
  "average_age": 34.5
}
{
  "_id": "财务",
  "average_age": 32.5
}
```

第8.7节：获取样本数据

要从某个集合中获取随机数据，请参考$sample聚合操作。

```
db.emplyees.aggregate({ $sample: { size:1 } })
```

其中 size表示要选择的项目数量。

第8.8节：删除集合中具有重复字段的文档（去重）

请注意，allowDiskUse: true选项是可选的，但它有助于缓解内存不足的问题，因为如果集合大小较大，此聚合操作可能会占用大量内存——因此我建议始终使用它。

```
var duplicates = [];

db.transactions.aggregate([
 { $group: {
_id: { cr_dr: "$cr_dr"},
   dups: { "$addToSet": "$_id" },
   count: { "$sum": 1 }
 }
},
{ $match: {
count: { "$gt": 1 }
}}
],allowDiskUse: true}
)
.result
.forEach(function(doc) {
  doc.dups.shift();
  doc.dups.forEach( function(dupId){
    duplicates.push(dupId);
  }
)
})
// printjson(duplicates);

// Remove all duplicates in one go
db.transactions.remove({_id:{$in:duplicates}})
```

第8.9节：带聚合的左外连接（$Lookup）

```
let col_1 = db.collection('col_1');
let col_2 = db.collection('col_2');
col_1 .aggregate([
    { $match: { "_id": 1 } },
    {
$lookup: {
            from: "col_2",
            localField: "id",
```



```
  "average_age": 34.5
}
{
  "_id": "Finance",
  "average_age": 32.5
}
```

Section 8.7: Get sample data

To get random data from certain collection refer to `$sample` aggregation.

```
db.emplyees.aggregate({ $sample: { size:1 } })
```

where `size` stands for number of items to select.

Section 8.8: Remove docs that have a duplicate field in a collection (dedupe)

Note that the allowDiskUse: true option is optional but will help mitigate out of memory issues as this aggregation can be a memory intensive operation if your collection size is large - so i recommend to always use it.

```
var duplicates = [];

db.transactions.aggregate([
 { $group: {
   _id: { cr_dr: "$cr_dr"},
   dups: { "$addToSet": "$_id" },
   count: { "$sum": 1 }
 }
},
{ $match: {
  count: { "$gt": 1 }
}}
],allowDiskUse: true}
)
.result
.forEach(function(doc) {
  doc.dups.shift();
  doc.dups.forEach( function(dupId){
    duplicates.push(dupId);
  }
)
})
// printjson(duplicates);

// Remove all duplicates in one go
db.transactions.remove({_id:{$in:duplicates}})
```

Section 8.9: Left Outer Join with aggregation ($Lookup)

```
let col_1 = db.collection('col_1');
let col_2 = db.collection('col_2');
col_1 .aggregate([
    { $match: { "_id": 1 } },
    {
        $lookup: {
            from: "col_2",
            localField: "id",
```

```
        foreignField: "id",
                as: "new_document"
            }
        }
],function (err, result){
    res.send(result);
});
```

此功能在 mongodb 版本 3.2 中新发布，允许用户将一个集合与另一个集合中匹配属性的文档进行关联

MongoDB $LookUp 文档

第 8.10 节：服务器聚合

Andrew Mao 的解决方案。Meteor 中的平均聚合查询

```
Meteor.publish("someAggregation", function (args) {
    var sub = this;
    // 这适用于 Meteor 0.6.5
    var db = MongoInternals.defaultRemoteCollectionDriver().mongo.db;

    // 你传给 Mongo 聚合的参数。可以根据需要自行设置。
    var pipeline = [
        { $match: doSomethingWith(args) },
        { $group: {
_id: whatWeAreGroupingWith(args),
            count: { $sum: 1 }
        }}
    ];

db.collection("server_collection_name").aggregate(
        pipeline,
        // 需要包装回调函数，使其在 Fiber 中调用。
Meteor.bindEnvironment(
                function(err, result) {
                    // 将每个结果添加到订阅中。
_.each(result, function(e) {
                        // 为聚合文档生成一个随机的一次性ID
                     sub.added("client_collection_name", Random.id(), {
                        key: e._id.somethingOfInterest,
                        count: e.count
                        });
                    });
sub.ready();
                },
                function(error) {
Meteor._debug( "执行聚合时出错: " + error);
                }
        )
    );
});
```

第8.11节：服务器方法中的聚合

另一种执行聚合的方法是使用`Mongo.Collection#rawCollection()`

这只能在服务器上运行。


```
            foreignField: "id",
            as: "new_document"
        }
    }
],function (err, result){
    res.send(result);
});
```

This feature was newly released in the mongodb **version 3.2** , which gives the user a stage to join one collection with the matching attributes from another collection

Mongodb $LookUp documentation

Section 8.10: Server Aggregation

Andrew Mao's solution. Average Aggregation Queries in Meteor

```
Meteor.publish("someAggregation", function (args) {
    var sub = this;
    // This works for Meteor 0.6.5
    var db = MongoInternals.defaultRemoteCollectionDriver().mongo.db;

    // Your arguments to Mongo's aggregation. Make these however you want.
    var pipeline = [
        { $match: doSomethingWith(args) },
        { $group: {
            _id: whatWeAreGroupingWith(args),
            count: { $sum: 1 }
        }}
    ];

    db.collection("server_collection_name").aggregate(
        pipeline,
        // Need to wrap the callback so it gets called in a Fiber.
        Meteor.bindEnvironment(
            function(err, result) {
                // Add each of the results to the subscription.
                _.each(result, function(e) {
                    // Generate a random disposable id for aggregated documents
                    sub.added("client_collection_name", Random.id(), {
                        key: e._id.somethingOfInterest,
                        count: e.count
                    });
                });
                sub.ready();
            },
            function(error) {
                Meteor._debug( "Error doing aggregation: " + error);
            }
        )
    );
});
```

Section 8.11: Aggregation in a Server Method

Another way of doing aggregations is by using the `Mongo.Collection#rawCollection()`

This can only be run on the Server.

以下是一个可用于Meteor 1.3及更高版本的示例：

```
Meteor.methods({
   'aggregateUsers'(someId) {
      const collection = MyCollection.rawCollection()
      const aggregate = Meteor.wrapAsync(collection.aggregate, collection)

      const match = { age: { $gte: 25 } }
      const group = { _id:'$age', totalUsers: { $sum: 1 } }

      const results = aggregate([
         { $match: match },
         { $group: group }
      ])

      return results
   }
})
```

第8.12节：Java和Spring示例

这是一个使用Spring Data在MongoDB中创建并执行聚合查询的示例代码。

```
    try {
MongoClient mongo = new MongoClient();
        DB db = mongo.getDB("so");
DBCollection coll = db.getCollection("employees");

        //等同于 $match
DBObject matchFields = new BasicDBObject();
        matchFields.put("dept", "Admin");
DBObject match = new BasicDBObject("$match", matchFields);

        //等同于 $project
DBObject projectFields = new BasicDBObject();
        projectFields.put("_id", 1);
projectFields.put("name", 1);
        projectFields.put("dept", 1);
        projectFields.put("totalExp", 1);
        projectFields.put("age", 1);
        projectFields.put("languages", 1);
        DBObject project = new BasicDBObject("$project", projectFields);

        //等同于 $group
DBObject groupFields = new BasicDBObject("_id", "$dept");
        groupFields.put("ageSet", new BasicDBObject("$addToSet", "$age"));
        DBObject employeeDocProjection = new BasicDBObject("$addToSet", new
BasicDBObject("totalExp", "$totalExp").append("age", "$age").append("languages",
"$languages").append("dept", "$dept").append("name", "$name"));
groupFields.put("docs", employeeDocProjection);
        DBObject group = new BasicDBObject("$group", groupFields);

        //按年龄排序结果
DBObject sort = new BasicDBObject("$sort", new BasicDBObject("age", 1));

        List<DBObject> aggregationList = new ArrayList<>();
aggregationList.add(match);
        aggregationList.add(project);
        aggregationList.add(group);
        aggregationList.add(sort);
```


Here is an example you can use in Meteor 1.3 and higher:

```
Meteor.methods({
   'aggregateUsers'(someId) {
      const collection = MyCollection.rawCollection()
      const aggregate = Meteor.wrapAsync(collection.aggregate, collection)

      const match = { age: { $gte: 25 } }
      const group = { _id:'$age', totalUsers: { $sum: 1 } }

      const results = aggregate([
         { $match: match },
         { $group: group }
      ])

      return results
   }
})
```

Section 8.12: Java and Spring example

This is an example code to create and execute the aggregate query in MongoDB using Spring Data.

```
    try {
        MongoClient mongo = new MongoClient();
        DB db = mongo.getDB("so");
        DBCollection coll = db.getCollection("employees");

        //Equivalent to $match
        DBObject matchFields = new BasicDBObject();
        matchFields.put("dept", "Admin");
        DBObject match = new BasicDBObject("$match", matchFields);

        //Equivalent to $project
        DBObject projectFields = new BasicDBObject();
        projectFields.put("_id", 1);
        projectFields.put("name", 1);
        projectFields.put("dept", 1);
        projectFields.put("totalExp", 1);
        projectFields.put("age", 1);
        projectFields.put("languages", 1);
        DBObject project = new BasicDBObject("$project", projectFields);

        //Equivalent to $group
        DBObject groupFields = new BasicDBObject("_id", "$dept");
        groupFields.put("ageSet", new BasicDBObject("$addToSet", "$age"));
        DBObject employeeDocProjection = new BasicDBObject("$addToSet", new
BasicDBObject("totalExp", "$totalExp").append("age", "$age").append("languages",
"$languages").append("dept", "$dept").append("name", "$name"));
        groupFields.put("docs", employeeDocProjection);
        DBObject group = new BasicDBObject("$group", groupFields);

        //Sort results by age
        DBObject sort = new BasicDBObject("$sort", new BasicDBObject("age", 1));

        List<DBObject> aggregationList = new ArrayList<>();
        aggregationList.add(match);
        aggregationList.add(project);
        aggregationList.add(group);
        aggregationList.add(sort);
```

```
AggregationOutput output = coll.aggregate(aggregationList);

        for (DBObject result : output.results()) {
            BasicDBList employeeList = (BasicDBList) result.get("docs");
            BasicDBObject employeeDoc = (BasicDBObject) employeeList.get(0);
            String name = employeeDoc.get("name").toString();
System.out.println(name);
        }
    }catch (Exception ex){
        ex.printStackTrace();
    }
```

查看 JSON 格式的 "resultSet" 值以了解输出格式：

```
[{
    "_id": "Admin",
    "ageSet": [35.0, 30.0],
    "docs": [{
        "totalExp": 11.0,
        "年龄": 35.0,
        "languages": ["english", "hindi"],
        "dept": "Admin",
        "name": "Anna"
    }, {
        "totalExp": 10.0,
        "age": 30.0,
        "languages": ["german", "french", "english", "hindi"],
        "dept": "Admin",
        "name": "Adma"
    }]
}]
```

"resultSet" 包含每个组的一个条目，"ageSet" 包含该组每个员工的年龄列表，"_id" 包含用于分组的字段值，"docs" 包含该组每个员工的数据，可用于我们自己的代码和用户界面。


```
        AggregationOutput output = coll.aggregate(aggregationList);

        for (DBObject result : output.results()) {
            BasicDBList employeeList = (BasicDBList) result.get("docs");
            BasicDBObject employeeDoc = (BasicDBObject) employeeList.get(0);
            String name = employeeDoc.get("name").toString();
            System.out.println(name);
        }
    }catch (Exception ex){
        ex.printStackTrace();
    }
```

See the "resultSet" value in JSON format to understand the output format:

```
[{
    "_id": "Admin",
    "ageSet": [35.0, 30.0],
    "docs": [{
        "totalExp": 11.0,
        "age": 35.0,
        "languages": ["english", "hindi"],
        "dept": "Admin",
        "name": "Anna"
    }, {
        "totalExp": 10.0,
        "age": 30.0,
        "languages": ["german", "french", "english", "hindi"],
        "dept": "Admin",
        "name": "Adma"
    }]
}]
```

The "resultSet" contains one entry for each group, "ageSet" contains the list of age of each employee of that group, "_id" contains the value of the field that is being used for grouping and "docs" contains data of each employee of that group that can be used in our own code and UI.

第9章：索引

第9.1节：索引创建基础

请参见下面的 transactions 集合。

```
> db.transactions.insert({ cr_dr : "D", amount : 100, fee : 2});
> db.transactions.insert({ cr_dr : "C", amount : 100, fee : 2});
> db.transactions.insert({ cr_dr : "C", amount : 10,  fee : 2});
> db.transactions.insert({ cr_dr : "D", amount : 100, fee : 4});
> db.transactions.insert({ cr_dr : "D", amount : 10,  fee : 2});
> db.transactions.insert({ cr_dr : "C", amount : 10,  fee : 4});
> db.transactions.insert({ cr_dr : "D", amount : 100, fee : 2});
```

getIndexes() 函数将显示集合中所有可用的索引。

```
db.transactions.getIndexes();
```

让我们看看上述语句的输出。

```
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    }
]
```

交易集合已经有一个索引。这是因为MongoDB在创建集合时，会在_id字段上创建一个唯一索引。该_id索引防止客户端插入两个具有相同_id字段值的文档。您不能删除_id字段上的此索引。

现在让我们为cr_dr字段添加一个索引；

```
db.transactions.createIndex({ cr_dr : 1 });
```

索引执行的结果如下。

```
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}
```

> createdCollectionAutomatically表示该操作是否创建了集合。如果集合不存在，MongoDB会在索引操作过程中创建该集合。

让我们再次运行 db.transactions.getIndexes();。

```
[
```


Chapter 9: Indexes

Section 9.1: Index Creation Basics

See the below transactions collection.

```
> db.transactions.insert({ cr_dr : "D", amount : 100, fee : 2});
> db.transactions.insert({ cr_dr : "C", amount : 100, fee : 2});
> db.transactions.insert({ cr_dr : "C", amount : 10,  fee : 2});
> db.transactions.insert({ cr_dr : "D", amount : 100, fee : 4});
> db.transactions.insert({ cr_dr : "D", amount : 10,  fee : 2});
> db.transactions.insert({ cr_dr : "C", amount : 10,  fee : 4});
> db.transactions.insert({ cr_dr : "D", amount : 100, fee : 2});
```

`getIndexes()` functions will show all the indices available for a collection.

```
db.transactions.getIndexes();
```

Let see the output of above statement.

```
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    }
]
```

There is already one index for transaction collection. This is because MongoDB creates a *unique index* on the `_id` field during the creation of a collection. The `_id` index prevents clients from inserting two documents with the same value for the `_id` field. You cannot drop this index on the `_id` field.

Now let's add an index for cr_dr field;

```
db.transactions.createIndex({ cr_dr : 1 });
```

The result of the index execution is as follows.

```
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}
```

> The createdCollectionAutomatically indicates if the operation created a collection. If a collection does not exist, MongoDB creates the collection as part of the indexing operation.

Let run `db.transactions.getIndexes();` again.

```
[
```

```
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : 1
        },
        "name" : "cr_dr_1",
        "ns" : "documentation_db.transactions"
    }
]
```

现在你看到 transactions 集合有两个索引。默认的 _id 索引和我们创建的 cr_dr_1。名称是由 MongoDB 分配的。你可以像下面这样设置自己的名称。

```
db.transactions.createIndex({ cr_dr : -1 },{name : "index on cr_dr desc"})
```

现在 db.transactions.getIndexes(); 会给你三个索引。

```
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : 1
        },
        "name" : "cr_dr_1",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : -1
        },
        "name" : "index on cr_dr desc",
        "ns" : "documentation_db.transactions"
    }
]
```

创建索引时 { cr_dr : -1 } 中 1 表示索引将是 升序，-1 表示 降序。

版本 ≥ 2.4

哈希索引

索引也可以定义为*哈希*索引。这在*等值查询*中性能更好，但对于*范围查询*效率不高；不过你可以在同一字段上同时定义哈希索引和升序/降序索引。


```
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : 1
        },
        "name" : "cr_dr_1",
        "ns" : "documentation_db.transactions"
    }
]
```

Now you see transactions collection have two indices. Default `_id` index and `cr_dr_1` which we created. The name is assigned by MongoDB. You can set your own name like below.

```
db.transactions.createIndex({ cr_dr : -1 },{name : "index on cr_dr desc"})
```

Now `db.transactions.getIndexes();` will give you three indices.

```
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : 1
        },
        "name" : "cr_dr_1",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : -1
        },
        "name" : "index on cr_dr desc",
        "ns" : "documentation_db.transactions"
    }
]
```

While creating index `{ cr_dr : -1 }` 1 means index will be in `ascending` order and -1 for `descending` order.

Version ≥ 2.4

Hashed indexes

Indexes can be defined also as *hashed*. This is more performant on *equality queries*, but is not efficient for *range queries*; however you can define both hashed and ascending/descending indexes on the same field.

```
> db.transactions.createIndex({ cr_dr : "hashed" });

> db.transactions.getIndexes(
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : "hashed"
        },
        "name" : "cr_dr_hashed",
        "ns" : "documentation_db.transactions"
    }
]
```

第9.2节：删除索引

如果已知索引名称，

```
db.collection.dropIndex('索引名称');
```

如果不知道索引名称，

```
db.collection.dropIndex( { '字段名称' : -1 } );
```

第9.3节：稀疏索引和部分索引

稀疏索引：

这些对于可选但也应唯一的字段特别有用。

```
{ "_id" : "john@example.com", "nickname" : "Johnnie" }
{ "_id" : "jane@example.com" }
{ "_id" : "julia@example.com", "nickname" : "Jules"}
{ "_id" : "jack@example.com" }
```

由于两个条目没有指定“nickname”，且索引会将未指定的字段视为null，索引创建
将因有2个文档的“null”而失败，因此：

```
db.scores.createIndex( { nickname: 1 } , { unique: true, sparse: true } )
```

将允许你仍然拥有“null”的昵称。

稀疏索引更紧凑，因为它们跳过/忽略未指定该字段的文档。因此，如果你的
集合中只有不到10%的文档指定了该字段，你可以创建更小的索引——如果你想执行如下查询，
这将更好地利用有限的内存：

```
db.scores.find({'nickname': 'Johnnie'})
```



```
> db.transactions.createIndex({ cr_dr : "hashed" });

> db.transactions.getIndexes(
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : "hashed"
        },
        "name" : "cr_dr_hashed",
        "ns" : "documentation_db.transactions"
    }
]
```

Section 9.2: Dropping/Deleting an Index

If index name is known,

```
db.collection.dropIndex('name_of_index');
```

If index name is not known,

```
db.collection.dropIndex( { 'name_of_field' : -1 } );
```

Section 9.3: Sparse indexes and Partial indexes

Sparse indexes:

These can be particularly useful for fields that are optional but which should also be unique.

```
{ "_id" : "john@example.com", "nickname" : "Johnnie" }
{ "_id" : "jane@example.com" }
{ "_id" : "julia@example.com", "nickname" : "Jules"}
{ "_id" : "jack@example.com" }
```

Since two entries have no "nickname" specified and indexing will treat unspecified fields as null, the index creation would fail with 2 documents having 'null', so:

```
db.scores.createIndex( { nickname: 1 } , { unique: true, sparse: true } )
```

will let you still have 'null' nicknames.

Sparse indexes are more compact since they skip/ignore documents that don't specify that field. So if you have a collection where only less than 10% of documents specify this field, you can create much smaller indexes - making better use of limited memory if you want to do queries like:

```
db.scores.find({'nickname': 'Johnnie'})
```

部分索引：

> 部分索引代表了稀疏索引功能的超集，应优先于稀疏索引使用。（3.2版本新增）

部分索引根据指定的筛选条件确定索引条目。

```
db.restaurants.createIndex(
  { cuisine: 1 },
  { partialFilterExpression: { rating: { $gt: 5 } } }
)
```

如果 rating 大于5，则 cuisine 将被索引。是的，我们也可以根据其他属性的值来指定要索引的属性。

稀疏索引（Sparse）和部分索引（Partial）之间的区别：

稀疏索引仅根据被索引字段的存在来选择要索引的文档，或者对于复合索引，则根据被索引字段的存在来选择。

部分索引根据指定的过滤条件来确定索引条目。过滤条件可以包含索引键以外的字段，并且可以指定不仅仅是存在性检查的条件。

不过，部分索引可以实现与稀疏索引相同的行为

例如：

```
db.contacts.createIndex(
   { name: 1 },
   { partialFilterExpression: { name: { $exists: true } } }
)
```

> 注意： partialFilterExpression 选项和 sparse 选项不能同时指定。

第9.4节：获取集合的索引

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

输出

```
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : 1
        },
```


Partial indexes:

> Partial indexes represent a superset of the functionality offered by sparse indexes and should be preferred over sparse indexes. (*New in version 3.2*)

Partial indexes determine the index entries based on the specified filter.

```
db.restaurants.createIndex(
  { cuisine: 1 },
  { partialFilterExpression: { rating: { $gt: 5 } } }
)
```

If `rating` is greater than 5, then `cuisine` will be indexed. Yes, we can specify a property to be indexed based on the value of other properties also.

Difference between Sparse and Partial indexes:

Sparse indexes select documents to index solely based on the existence of the indexed field, or for compound indexes, the existence of the indexed fields.

Partial indexes determine the index entries based on the specified filter. The filter can include fields other than the index keys and can specify conditions other than just an existence check.

Still, a partial index can implement the same behavior as a sparse index

Eg:

```
db.contacts.createIndex(
   { name: 1 },
   { partialFilterExpression: { name: { $exists: true } } }
)
```

> **Note:** Both the *partialFilterExpression* option and the *sparse* option cannot be specified at the same time.

Section 9.4: Get Indices of a Collection

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

Output

```
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : 1
        },
```

```
        "name" : "cr_dr_1",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : -1
        },
        "name" : "index on cr_dr desc",
        "ns" : "documentation_db.transactions"
    }
]
```

第9.5节：复合索引

```
db.people.createIndex({name: 1, age: -1})
```

这将在多个字段上创建索引，在本例中是对name和age字段创建索引。它将在name字段上升序，在age字段降序。

在这种类型的索引中，排序顺序很重要，因为它决定了索引是否能支持排序操作。只要排序方向对所有排序键都是反向排序，复合索引的任何前缀都支持反向排序。否则，复合索引的排序必须匹配索引的顺序。

字段顺序也很重要，在本例中，索引首先按name排序，在每个name值内，再按age字段的值排序。这允许索引被用于对name字段的查询，或对name和age的查询，但不能单独用于age字段的查询。

第9.6节：唯一索引

```
db.collection.createIndex( { "user_id": 1 }, { unique: true } )
```

对定义的索引（单字段或复合索引）强制唯一性。如果集合中已经包含重复值，构建索引将失败；如果有多个条目缺少该字段，索引也会失败（因为它们都会被索引为值null），除非指定了 sparse: true。

第9.7节：单字段

```
db.people.createIndex({name: 1})
```

这将在字段name上创建一个升序的单字段索引。

在这种类型的索引中，排序顺序无关紧要，因为Mongo可以双向遍历索引。

第9.8节：删除

要删除索引，可以使用索引名称

```
db.people.dropIndex("nameIndex")
```

或者使用索引规范文档

```
db.people.dropIndex({name: 1})
```



```
        "name" : "cr_dr_1",
        "ns" : "documentation_db.transactions"
    },
    {
        "v" : 1,
        "key" : {
            "cr_dr" : -1
        },
        "name" : "index on cr_dr desc",
        "ns" : "documentation_db.transactions"
    }
]
```


Section 9.5: Compound

```
db.people.createIndex({name: 1, age: -1})
```

This creates an index on multiple fields, in this case on the `name` and `age` fields. It will be ascending in `name` and descending in `age`.

In this type of index, the sort order is relevant, because it will determine whether the index can support a sort operation or not. Reverse sorting is supported on any prefix of a compound index, as long as the sort is in the reverse sort direction for **all** of the keys in the sort. Otherwise, sorting for compound indexes need to match the order of the index.

Field order is also important, in this case the index will be sorted first by `name`, and within each name value, sorted by the values of the `age` field. This allows the index to be used by queries on the `name` field, or on `name` and `age`, but not on `age` alone.

Section 9.6: Unique Index

```
db.collection.createIndex( { "user_id": 1 }, { unique: true } )
```

enforce uniqueness on the defined index (either single or compound). Building the index will fail if the collection already contains duplicate values; the indexing will fail also with multiple entries missing the field (since they will all be indexed with the value **`null`**) unless `sparse: true` is specified.

Section 9.7: Single field

```
db.people.createIndex({name: 1})
```

This creates an ascending single field index on the field *name*.

In this type of indexes the sort order is irrelevant, because mongo can traverse the index in both directions.

Section 9.8: Delete

To drop an index you could use the index name

```
db.people.dropIndex("nameIndex")
```

Or the index specification document

```
db.people.dropIndex({name: 1})
```

第9.9节：列出

```
db.people.getIndexes()
```

这将返回一个文档数组，每个文档描述了people集合上的一个索引

Section 9.9: List

```
db.people.getIndexes()
```

This will return an array of documents each describing an index on the *people* collection

第10章：批量操作

第10.1节：将字段转换为另一种类型并批量更新整个集合

通常情况下，当需要将字段类型更改为另一种类型时，例如原始集合中可能将“数字”或“日期”字段保存为字符串：

```
{
    "name": "Alice",
    "salary": "57871",
    "dob": "1986-08-21"
},
{
    "name": "Bob",
    "salary": "48974",
    "dob": "1990-11-04"
}
```

目标是将像上面这样庞大的集合更新为

```
{
    "name": "Alice",
    "salary": 57871,
    "dob": ISODate("1986-08-21T00:00:00.000Z")
},
{
    "name": "鲍勃",
    "salary": 48974,
    "dob": ISODate("1990-11-04T00:00:00.000Z")
}
```

对于相对较小的数据，可以通过使用带有游标的**快照(snapshot)**和`forEach()`方法迭代集合，并按如下方式更新每个文档来实现上述操作：

```
db.test.find({
    "salary": { "$exists": true, "$type": 2 },
    "dob": { "$exists": true, "$type": 2 }
}).snapshot().forEach(function(doc){
    var newSalary = parseInt(doc.salary),
        newDob = new ISODate(doc.dob);
    db.test.updateOne(
        { "_id": doc._id },
        { "$set": { "salary": newSalary, "dob": newDob } }
    );
});
```

虽然这对于小型集合是最优的，但对于大型集合性能会大幅下降，因为遍历大量数据集并且每次请求都发送更新操作会带来计算开销。

Bulk() API 解决了这个问题，大大提升了性能，因为写操作只需一次性批量发送到服务器。效率的提升在于该方法不像当前在forEach()循环中那样每次写请求都发送到服务器，而是每1000个请求只发送一次，从而使更新比当前方式更高效、更快速。

Chapter 10: Bulk Operations

Section 10.1: Converting a field to another type and updating the entire collection in Bulk

Usually the case when one wants to change a field type to another, for instance the original collection may have "numerical" or "date" fields saved as strings:

```
{
    "name": "Alice",
    "salary": "57871",
    "dob": "1986-08-21"
},
{
    "name": "Bob",
    "salary": "48974",
    "dob": "1990-11-04"
}
```

The objective would be to update a humongous collection like the above to

```
{
    "name": "Alice",
    "salary": 57871,
    "dob": ISODate("1986-08-21T00:00:00.000Z")
},
{
    "name": "Bob",
    "salary": 48974,
    "dob": ISODate("1990-11-04T00:00:00.000Z")
}
```

For relatively small data, one can achieve the above by iterating the collection using a `snapshot` with the cursor's `forEach()` method and updating each document as follows:

```
db.test.find({
    "salary": { "$exists": true, "$type": 2 },
    "dob": { "$exists": true, "$type": 2 }
}).snapshot().forEach(function(doc){
    var newSalary = parseInt(doc.salary),
        newDob = new ISODate(doc.dob);
    db.test.updateOne(
        { "_id": doc._id },
        { "$set": { "salary": newSalary, "dob": newDob } }
    );
});
```

Whilst this is optimal for small collections, performance with large collections is greatly reduced since looping through a large dataset and sending each update operation per request to the server incurs a computational penalty.

The `Bulk()` API comes to the rescue and greatly improves performance since write operations are sent to the server only once in bulk. Efficiency is achieved since the method does not send every write request to the server (as with the current update statement within the `forEach()` loop) but just once in every 1000 requests, thus making updates more efficient and quicker than currently is.

使用上述相同的概念，通过forEach()循环来创建批处理，我们可以批量更新集合如下。在本演示中，MongoDB版本>= 2.6且< 3.2中可用的Bulk() API使用initializeUnorderedBulkOp()方法以并行且非确定性顺序执行批处理中的写入操作。

它通过将 salary和 dob字段分别更改为数值和日期时间值，更新clients集合中的所有文档：

```
var bulk = db.test.initializeUnorderedBulkOp(),
    counter = 0; // 用于跟踪批量更新大小的计数器

db.test.find({
     "salary": { "$exists": true, "$type": 2 },
     "dob": { "$exists": true, "$type": 2 }
}).snapshot().forEach(function(doc){
     var newSalary = parseInt(doc.salary),
newDob = new ISODate(doc.dob);
bulk.find({ "_id": doc._id }).updateOne({
         "$set": { "salary": newSalary, "dob": newDob }
     });

counter++; // 计数器加一
     if (counter % 1000 == 0) {
bulk.execute(); // 每1000次操作执行一次，并在每1000次更新语句后重新初始化

bulk = db.test.initializeUnorderedBulkOp();
     }
});
```

以下示例适用于新版 MongoDB 3.2，该版本已弃用 Bulk() API，改用 bulkWrite() 提供一组新的 API。

它使用与上述相同的游标，但通过相同的 forEach() 游标方法创建包含批量操作的数组，将每个批量写入文档推入数组。由于写入命令最多只能接受1000个操作，因此需要将操作分组，每组最多1000个操作，并在循环达到1000次迭代时重新初始化数组：

```
var cursor = db.test.find({
         "salary": { "$exists": true, "$type": 2 },
         "dob": { "$exists": true, "$type": 2 }
    }),
bulkUpdateOps = [];

cursor.snapshot().forEach(function(doc){
    var newSalary = parseInt(doc.salary),
        newDob = new ISODate(doc.dob);
    bulkUpdateOps.push({
        "updateOne": {
            "filter": { "_id": doc._id },
            "update": { "$set": { "salary": newSalary, "dob": newDob } }
         }
    });

    if (bulkUpdateOps.length === 1000) {
        db.test.bulkWrite(bulkUpdateOps);
        bulkUpdateOps = [];
    }
});
```


Using the same concept above with the `forEach()` loop to create the batches, we can update the collection in bulk as follows. In this demonstration the `Bulk()` API available in MongoDB versions `>= 2.6` and `< 3.2` uses the `initializeUnorderedBulkOp()` method to execute in parallel, as well as in a nondeterministic order, the write operations in the batches.

It updates all the documents in the clients collection by changing the `salary` and `dob` fields to `numerical` and `datetime` values respectively:

```
var bulk = db.test.initializeUnorderedBulkOp(),
    counter = 0; // counter to keep track of the batch update size

db.test.find({
    "salary": { "$exists": true, "$type": 2 },
    "dob": { "$exists": true, "$type": 2 }
}).snapshot().forEach(function(doc){
    var newSalary = parseInt(doc.salary),
        newDob = new ISODate(doc.dob);
    bulk.find({ "_id": doc._id }).updateOne({
        "$set": { "salary": newSalary, "dob": newDob }
    });

    counter++; // increment counter
    if (counter % 1000 == 0) {
        bulk.execute(); // Execute per 1000 operations and re-initialize every 1000 update
statements
        bulk = db.test.initializeUnorderedBulkOp();
    }
});
```

The next example applies to the new MongoDB version `3.2` which has since deprecated the `Bulk()` API and provided a newer set of apis using `bulkWrite()`.

It uses the same cursors as above but creates the arrays with the bulk operations using the same `forEach()` cursor method to push each bulk write document to the array. Because write commands can accept no more than 1000 operations, there's need to group operations to have at most 1000 operations and re-intialise the array when the loop hits the 1000 iteration:

```
var cursor = db.test.find({
        "salary": { "$exists": true, "$type": 2 },
        "dob": { "$exists": true, "$type": 2 }
    }),
    bulkUpdateOps = [];

cursor.snapshot().forEach(function(doc){
    var newSalary = parseInt(doc.salary),
        newDob = new ISODate(doc.dob);
    bulkUpdateOps.push({
        "updateOne": {
            "filter": { "_id": doc._id },
            "update": { "$set": { "salary": newSalary, "dob": newDob } }
         }
    });

    if (bulkUpdateOps.length === 1000) {
        db.test.bulkWrite(bulkUpdateOps);
        bulkUpdateOps = [];
    }
});
```

```
if (bulkUpdateOps.length > 0) { db.test.bulkWrite(bulkUpdateOps); }
```



```
if (bulkUpdateOps.length > 0) { db.test.bulkWrite(bulkUpdateOps); }
```

第11章：2dsphere 索引

第11.1节：创建2dsphere索引

db.collection.createIndex() 方法用于创建一个 2dsphere 索引。2dsphere 索引的结构如下：

```
db.collection.createIndex( { <位置字段> : "2dsphere" } )
```

这里，位置字段是键，2dsphere是索引的类型。在下面的示例中，我们将在places集合中创建一个2dsphere索引。

```
db.places.insert(
{
loc : { type: "Point", coordinates: [ -73.97, 40.77 ] },
   name: "中央公园",
category : "公园"
})
```

以下操作将在places集合的loc字段上创建2dsphere索引。

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


Chapter 11: 2dsphere Index

Section 11.1: Create a 2dsphere Index

`db.collection.createIndex()` method is used to create a `2dsphere` index. The blueprint of a `2dsphere` index :

```
db.collection.createIndex( { <location field> : "2dsphere" } )
```

Here, the `location field` is the key and `2dsphere` is the type of the index. In the following example we are going to create a `2dsphre` index in the `places` collection.

```
db.places.insert(
{
  loc : { type: "Point", coordinates: [ -73.97, 40.77 ] },
  name: "Central Park",
  category : "Parks"
})
```

The following operation will create `2dsphere` index on the `loc` field of `places` collection.

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

第12章：可插拔存储引擎

第12.1节：WiredTiger

WiredTiger支持LSM树来存储索引。LSM树在需要写入大量随机插入的工作负载时，写操作更快。

在WiredTiger中，没有原地更新。如果需要更新文档的某个元素，将插入一个新文档，同时删除旧文档。

WiredTiger还提供文档级并发。它假设两个写操作不会影响同一文档，但如果发生，某个操作将被回滚并稍后执行。如果回滚很少见，这将极大提升性能。

WiredTiger支持Snappy和zLib算法对文件系统中的数据和索引进行压缩。默认使用Snappy。它对CPU的消耗较低，但压缩率低于zLib。

如何使用 WiredTiger 引擎

```
mongod --storageEngine wiredTiger --dbpath <newWiredTigerDBPath>
```

注意：

1. 在 mongodb 3.2 之后，默认引擎是 WiredTiger。
2. newWiredTigerDBPath 不应包含其他存储引擎的数据。要迁移数据，您必须导出数据，然后在新的存储引擎中重新导入。

```
mongodump --out <exportDataDestination>
    mongod --storageEngine wiredTiger --dbpath <newWiredTigerDBPath>
    mongorestore <exportDataDestination>
```

第12.2节：MMAP

MMAP 是一种可插拔的存储引擎，名称来源于 Linux 命令 mmap()。它将文件映射到虚拟内存并优化读取调用。如果您有一个大文件但只需要读取其中一小部分，mmap() 比 read() 调用要快得多，后者会将整个文件加载到内存中。

一个缺点是你不能对同一个集合同时处理两个写操作。因此，MMAP 采用集合级锁定（而不是 WiredTiger 提供的文档级锁定）。这种集合锁定是必要的，因为一个 MMAP 索引可以引用多个文档，如果这些文档能够同时被更新，索引将会不一致。

第12.3节：内存中

所有数据都存储在内存（RAM）中，以实现更快的读取/访问。

第12.4节：mongo-rocks

一个为集成Facebook的RocksDB而创建的键值引擎。

第12.5节：Fusion-io

由SanDisk创建的存储引擎，可以绕过操作系统文件系统层，直接写入存储设备。

Chapter 12: Pluggable Storage Engines

Section 12.1: WiredTiger

WiredTiger supports **LSM trees to store indexes**. LSM trees are faster for write operations when you need to write huge workloads of random inserts.

In WiredTiger, there is **no in-place updates**. If you need to update an element of a document, a new document will be inserted while the old document will be deleted.

WiredTiger also offers **document-level concurrency**. It assumes that two write operations will not affect the same document, but if it does, one operation will be rewind and executed later. That's a great performance boost if rewinds are rare.

WiredTiger supports **Snappy and zLib algorithms for compression** of data and indexes in the file system. Snappy is the default. It is less CPU-intensive but have a lower compression rate than zLib.

How to use WiredTiger Engine

```
mongod --storageEngine wiredTiger --dbpath <newWiredTigerDBPath>
```

Note:

1. After mongodb 3.2, the default engine is WiredTiger.
2. `newWiredTigerDBPath` should not contain data of another storage engine. To migrate your data, you have to dump them, and re-import them in the new storage engine.

```
    mongodump --out <exportDataDestination>
    mongod --storageEngine wiredTiger --dbpath <newWiredTigerDBPath>
    mongorestore <exportDataDestination>
```

Section 12.2: MMAP

MMAP is a pluggable storage engine that was named after the `mmap()` Linux command. It maps files to the virtual memory and optimizes read calls. If you have a large file but needs to read just a small part of it, `mmap()` is much faster then a `read()` call that would bring the entire file to the memory.

One disadvantage is that you can't have two write calls being processed in parallel for the same collection. So, MMAP has collection-level locking (and not document-level locking as WiredTiger offers). This collection-locking is necessary because one MMAP index can reference multiples documents and if those docs could be updated simultaneously, the index would be inconsistent.

Section 12.3: In-memory

All data is stored in-memory (RAM) for faster read/access.

Section 12.4: mongo-rocks

A key-value engine created to integrate with Facebook's RocksDB.

Section 12.5: Fusion-io

A storage engine created by SanDisk that makes it possible to bypass the OS file system layer and write directly to

存储设备。

第12.6节：TokuMX

由Percona创建的存储引擎，使用分形树索引。

the storage device.

Section 12.6: TokuMX

A storage engine created by Percona that uses fractal tree indexes.

第13章：Java驱动程序

第13.1节：带条件获取集合数据

从 testdb 数据库中的 testcollection 集合获取 name = dev 的数据

```
import org.bson.Document;
import com.mongodb.BasicDBObject;
import com.mongodb.MongoClient;
import com.mongodb.ServerAddress;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;

MongoClient mongoClient = new MongoClient(new ServerAddress("localhost", 27017));
MongoDatabase db = mongoClient.getDatabase("testdb");
MongoCollection<Document> collection = db.getCollection("testcollection");

BasicDBObject searchQuery = new BasicDBObject();
searchQuery.put("name","dev");

MongoCursor<Document> cursor = collection.find(searchQuery).iterator();
try {
while (cursor.hasNext()) {
        System.out.println(cursor.next().toJson());
    }
} finally {
    cursor.close();
}
```

第13.2节：创建数据库用户

创建用户名为dev，密码为password123的用户

```
MongoClient mongo = new MongoClient("localhost", 27017);
MongoDatabase db =  mongo.getDatabase("testDB");
Map<String, Object> commandArguments = new BasicDBObject();
commandArguments.put("createUser", "dev");
commandArguments.put("pwd", "password123");
String[] roles = { "readWrite" };
commandArguments.put("roles", roles);
BasicDBObject command = new BasicDBObject(commandArguments);
db.runCommand(command);
```

第13.3节：创建可尾随游标

```
find(query).projection(fields).cursorType(CursorType.TailableAwait).iterator();
```

该代码适用于MongoCollection类。

CursorType 是一个枚举类型，具有以下值：

```
Tailable
TailableAwait
```

对应旧版（<3.0）DBCursor 的 addOption 字节类型：

Chapter 13: Java Driver

Section 13.1: Fetch Collection data with condition

To get data from `testcollection` collection in `testdb` database where `name=dev`

```
import org.bson.Document;
import com.mongodb.BasicDBObject;
import com.mongodb.MongoClient;
import com.mongodb.ServerAddress;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;

MongoClient mongoClient = new MongoClient(new ServerAddress("localhost", 27017));
MongoDatabase db = mongoClient.getDatabase("testdb");
MongoCollection<Document> collection = db.getCollection("testcollection");

BasicDBObject searchQuery = new BasicDBObject();
searchQuery.put("name","dev");

MongoCursor<Document> cursor = collection.find(searchQuery).iterator();
try {
    while (cursor.hasNext()) {
        System.out.println(cursor.next().toJson());
    }
} finally {
    cursor.close();
}
```

Section 13.2: Create a database user

To create a user **dev** with password **password123**

```
MongoClient mongo = new MongoClient("localhost", 27017);
MongoDatabase db =  mongo.getDatabase("testDB");
Map<String, Object> commandArguments = new BasicDBObject();
commandArguments.put("createUser", "dev");
commandArguments.put("pwd", "password123");
String[] roles = { "readWrite" };
commandArguments.put("roles", roles);
BasicDBObject command = new BasicDBObject(commandArguments);
db.runCommand(command);
```

Section 13.3: Create a tailable cursor

```
find(query).projection(fields).cursorType(CursorType.TailableAwait).iterator();
```

That code applies to the `MongoCollection` class.

CursorType is an enum and it has the following values:

```
Tailable
TailableAwait
```

Corresponding to the old (<3.0) DBCursor addOption Bytes types:

```
Bytes.QUERYOPTION_TAILABLE
Bytes.QUERYOPTION_AWAITDATA
```



```
Bytes.QUERYOPTION_TAILABLE
Bytes.QUERYOPTION_AWAITDATA
```

第14章：Python驱动程序

参数	详细信息
hostX	可选。您可以指定任意数量的主机。例如， 连接到副本集时可以指定多个主机。
:portX	可选。如果未指定，默认值为：27017。
数据库	可选。如果连接字符串包含身份验证，则为要进行身份验证的数据库名称 如果未指定 /database 且连接字符串包含凭据，驱动程序将 对管理员数据库进行身份验证。
?选项	连接特定选项

第14.1节：使用pymongo连接MongoDB

```
from pymongo import MongoClient

uri = "mongodb://localhost:27017/"

client = MongoClient(uri)

db = client['test_db']
# 或者
# db = client.test_db

# collection = db['test_collection']
# 或者
collection = db.test_collection


collection.save({"hello":"world"})

print collection.find_one()
```

第14.2节：PyMongo查询

一旦你获得了一个collection对象，查询语法与mongo shell中相同。有一些细微的区别：

- 每个键必须用引号括起来。例如：

```
db.find({frequencies: {$exists: true}})
```

在pymongo中变为（注意True首字母大写）：

```
db.find({"frequencies": { "$exists": True }})
```

- 诸如对象ID或ISODate之类的对象使用Python类进行操作。PyMongo使用其自己的ObjectId类来处理对象ID，而日期则使用标准的datetime包。例如，如果你想查询2010年到2011年之间的所有事件，可以这样做：

```
from datetime import datetime

date_from = datetime(2010, 1, 1)
date_to = datetime(2011, 1, 1)
db.find({ "date": { "$gte": date_from, "$lt": date_to } }):
```


Chapter 14: Python Driver

Parameter	Detail
hostX	Optional. You can specify as many hosts as necessary. You would specify multiple hosts, for example, for connections to replica sets.
:portX	Optional. The default value is :27017 if not specified.
database	Optional. The name of the database to authenticate if the connection string includes authentication credentialsIf /database is not specified and the connection string includes credentials, the driver will authenticate to the admin database.
?options	Connection specific options

Section 14.1: Connect to MongoDB using pymongo

```
from pymongo import MongoClient

uri = "mongodb://localhost:27017/"

client = MongoClient(uri)

db = client['test_db']
# or
# db = client.test_db

# collection = db['test_collection']
# or
collection = db.test_collection


collection.save({"hello":"world"})

print collection.find_one()
```

Section 14.2: PyMongo queries

Once you got a `collection` object, queries use the same syntax as in the mongo shell. Some slight differences are:

- every key must be enclosed in brackets. For example:

```
db.find({frequencies: {$exists: true}})
```

becomes in `pymongo` (note the `True` in uppercase):

```
db.find({"frequencies": { "$exists": True }})
```

- objects such as object ids or `ISODate` are manipulated using python classes. PyMongo uses its own `ObjectId` class to deal with object ids, while dates use the standard `datetime` package. For example, if you want to query all events between 2010 and 2011, you can do:

```
from datetime import datetime

date_from = datetime(2010, 1, 1)
date_to = datetime(2011, 1, 1)
db.find({ "date": { "$gte": date_from, "$lt": date_to } }):
```

第14.3节：使用

PyMongo更新集合中的所有文档

假设你需要向集合中的每个文档添加一个字段。

```
import pymongo

client = pymongo.MongoClient('localhost', 27017)
db = client.mydb.mycollection

for doc in db.find():
db.update(
        {'_id': doc['_id']},
        {'$set': {'newField': 10} }, upsert=False, multi=False)
```

find 方法返回一个Cursor，可以使用for in语法轻松迭代。然后，我们调用update方法，指定_id并添加一个字段（$set）。参数upsert和multi来自mongodb（更多信息见此）。

Section 14.3: Update all documents in a collection using PyMongo

Let's say you need to add a field to every document in a collection.

```
import pymongo

client = pymongo.MongoClient('localhost', 27017)
db = client.mydb.mycollection

for doc in db.find():
   db.update(
       {'_id': doc['_id']},
       {'$set': {'newField': 10} }, upsert=False, multi=False)
```

The `find` method returns a `Cursor`, on which you can easily iterate over using the **`for in`** syntax. Then, we call the `update` method, specifying the `_id` and that we add a field (`$set`). The parameters `upsert` and `multi` come from mongodb (see here for more info).

第15章：Mongo作为分片

第15.1节：分片环境设置

分片组成员：

分片有三个参与者。

1. 配置服务器
2. 副本集
3. Mongos

   对于一个Mongo分片，我们需要设置上述三台服务器。

配置服务器设置：在mongod配置文件中添加以下内容

```
sharding:
clusterRole: configsvr
replication:
replSetName: <setname>
```

运行： mongod --config

我们可以选择将配置服务器设置为副本集，也可以是独立服务器。根据需求选择最合适的。如果配置服务器需要以副本集方式运行，则需要按照副本集的设置进行配置。

副本集设置： 创建副本集 // 请参考副本集设置

MongoS设置： Mongos是分片中的主要设置。它是访问所有副本集的查询路由器

在mongos配置文件中添加以下内容

```
分片:
configDB: <configReplSetName>/cfg1.example.net:27017;
```

配置共享：

通过 shell 连接 mongos（mongo --host --port）

```
1.sh.addShard( "/s1-mongo1.example.net:27017")
2.sh.enableSharding("")
3.sh.shardCollection("<数据库>.<集合>", { <键> : <方向> } )
4.sh.status() // 确认分片状态
```


Chapter 15: Mongo as Shards

Section 15.1: Sharding Environment Setup

Sharding Group Members :

For sharding there are three players.

1. Config Server
2. Replica Sets
3. Mongos

   For a mongo shard we need to setup the above three servers.

Config Server Setup : add the following to mongod conf file

```
sharding:
  clusterRole: configsvr
replication:
  replSetName: <setname>
```

run : mongod --config

we can choose config server as replica set or may be a standalone server. Based on our requirement we can choose the best. If config need to run in replica set we need to follow the replica set setup

Replica Setup : Create replica set // Please refer the replica setup

MongoS Setup : Mongos is main setup in shard. Its is query router to access all replica sets

Add the following in mongos conf file

```
  sharding:
    configDB: <configReplSetName>/cfg1.example.net:27017;
```

Configure Shared :

Connect the mongos via shell (mongo --host --port)

1. sh.addShard("/s1-mongo1.example.net:27017")
2. sh.enableSharding("")
3. sh.shardCollection("< database >.< collection >", { < key > : < direction > })
4. sh.status() // To ensure the sharding

第16章：复制

第16.1节：三节点的基本配置

副本集是一组维护相同数据集的 mongod 实例。

本示例展示如何在同一服务器上配置包含三个实例的副本集。

创建数据文件夹

```
mkdir /srv/mongodb/data/rs0-0
mkdir /srv/mongodb/data/rs0-1
mkdir /srv/mongodb/data/rs0-2
```

启动 mongod 实例

```
mongod --port 27017 --dbpath /srv/mongodb/data/rs0-0 --replSet rs0
mongod --port 27018 --dbpath /srv/mongodb/data/rs0-1 --replSet rs0
mongod --port 27019 --dbpath /srv/mongodb/data/rs0-2 --replSet rs0
```

配置副本集

```
mongo --port 27017    // 连接到端口27017的实例

rs.initiate();                // 在第一个节点初始化副本集
rs.add("<hostname>:27018")    // 添加第二个节点
rs.add("<hostname>:27019")    // 添加第三个节点
```

测试您的设置

检查配置时输入 rs.status()，结果应如下所示：

```
{
        "set" : "rs0",
        "date" : ISODate("2016-09-01T12:34:24.968Z"),
        "myState" : 1,
        "term" : NumberLong(4),
        "heartbeatIntervalMillis" : NumberLong(2000),
        "members" : [
                {
                        "_id" : 0,
                        "name" : "<hostname>:27017",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        .........................
                },
                {
                        "_id" : 1,
                        "name" : "<hostname>:27018",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        .........................
                },
                {
                        "_id" : 2,
                        "name" : "<hostname>:27019",
```


Chapter 16: Replication

Section 16.1: Basic configuration with three nodes

The replica set is a group of mongod instances that maintain the same data set.

This example shows how to configure a replica set with three instances on the same server.

Creating data folders

```
mkdir /srv/mongodb/data/rs0-0
mkdir /srv/mongodb/data/rs0-1
mkdir /srv/mongodb/data/rs0-2
```

Starting mongod instances

```
mongod --port 27017 --dbpath /srv/mongodb/data/rs0-0 --replSet rs0
mongod --port 27018 --dbpath /srv/mongodb/data/rs0-1 --replSet rs0
mongod --port 27019 --dbpath /srv/mongodb/data/rs0-2 --replSet rs0
```

Configuring replica set

```
mongo --port 27017    // connection to the instance 27017

rs.initiate();                // initilization of replica set on the 1st node
rs.add("<hostname>:27018")    // adding a 2nd node
rs.add("<hostname>:27019")    // adding a 3rd node
```

Testing your setup

For checking the configuration type `rs.status()`, the result should be like:

```
{
        "set" : "rs0",
        "date" : ISODate("2016-09-01T12:34:24.968Z"),
        "myState" : 1,
        "term" : NumberLong(4),
        "heartbeatIntervalMillis" : NumberLong(2000),
        "members" : [
                {
                        "_id" : 0,
                        "name" : "<hostname>:27017",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        .........................
                },
                {
                        "_id" : 1,
                        "name" : "<hostname>:27018",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        .........................
                },
                {
                        "_id" : 2,
                        "name" : "<hostname>:27019",
```

```
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        ..........................
                }
        ],
        "ok" : 1
}
```



```
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        ..........................
                }
        ],
        "ok" : 1
}
```

第17章：Mongo作为副本集

第17.1节：Mongodb作为副本集

我们将创建一个包含3个实例的mongodb副本集。一个实例将作为主节点，另外两个实例将作为从节点。

为了简化，我将使用同一台服务器上运行的3个mongodb实例组成副本集，
为此，三个mongodb实例将运行在不同的端口号上。

在生产环境中，如果你有一个专用的 MongoDB 实例运行在单台服务器上，你可以重复使用相同的端口号。

1. 创建数据目录（mongodb 数据存储的文件路径）

```
-mkdir c:\data\server1 （实例 1 的数据文件路径）
-mkdir c:\data\server2 （实例 2 的数据文件路径）
-mkdir c:\data\server3 （实例 3 的数据文件路径）
```

2.a. 启动第一个 mongod 实例

- 打开命令提示符，输入以下命令并按回车。

```
mongod --replSet s0 --dbpath c:\data\server1 --port 37017  --smallfiles --oplogSize 100
```

上述命令将 mongodb 实例关联到副本集名称“s0”，并在端口 37017 上启动第一个 mongodb 实例，oplogSize 为 100MB

2.b. 同样启动第二个 Mongodb 实例

```
mongod --replSet s0 --dbpath c:\data\server2 --port 37018  --smallfiles --oplogSize 100
```

上述命令将mongodb实例关联到名为"s0"的副本集，并启动第一个mongodb实例
，端口为37018，oplogSize为100MB

2.c. 现在启动第三个Mongodb实例

```
mongod --replSet s0 --dbpath c:\data\server3 --port 37019  --smallfiles --oplogSize 100
```

上述命令将mongodb实例关联到名为"s0"的副本集，并启动第一个mongodb实例
，端口为37019，oplogSize为100MB

当三个实例都启动后，这三个实例目前彼此独立。我们现在需要将这些实例分组为一个副本集。我们通过配置对象来实现这一点。

3.a 通过mongo shell连接到任意一个mongod服务器。为此，打开命令提示符并输入。

```
mongo --port 37017
```

连接到mongo shell后，创建一个配置对象

```
    var config = {"_id":"s0", members[]};
```

该配置对象有两个属性

Chapter 17: Mongo as a Replica Set

Section 17.1: Mongodb as a Replica Set

We would be creating mongodb as a replica set having 3 instances. One instance would be primary and the other 2 instances would be secondary.

For simplicity, I am going to have a replica set with 3 instances of mongodb running on the same server and thus to achieve this, all three mongodb instances would be running on different port numbers.

In production environment where in you have a dedicated mongodb instance running on a single server you can reuse the same port numbers.

1. Create data directories (path where mongodb data would be stored in a file)

```
- mkdir c:\data\server1 (datafile path for instance 1)
- mkdir c:\data\server2 (datafile path for instance 2)
- mkdir c:\data\server3 (datafile path for instance 3)
```

2. a. Start the first mongod instance

- Open command prompt and type the following press enter.

```
mongod --replSet s0 --dbpath c:\data\server1 --port 37017  --smallfiles --oplogSize 100
```

The above command associates the instance of mongodb to a replicaSet name "s0" and the starts the first instance of mongodb on port 37017 with oplogSize 100MB

2. b. Similarly start the second instance of Mongodb

```
mongod --replSet s0 --dbpath c:\data\server2 --port 37018  --smallfiles --oplogSize 100
```

The above command associates the instance of mongodb to a replicaSet name "s0" and the starts the first instance of mongodb on port 37018 with oplogSize 100MB

2. c. Now start the third instance of Mongodb

```
mongod --replSet s0 --dbpath c:\data\server3 --port 37019  --smallfiles --oplogSize 100
```

The above command associates the instance of mongodb to a replicaSet name "s0" and the starts the first instance of mongodb on port 37019 with oplogSize 100MB

With all the 3 instances started, these 3 instances are independent of each other currently. We would now need to group these instances as a replica set. We do this with the help of a config object.

3.a Connect to any of the mongod servers via the mongo shell. To do that open the command prompt and type.

```
mongo --port 37017
```

Once connected to the mongo shell, create a config object

```
   var config = {"_id":"s0", members[]};
```

this config object has 2 attributes

1._id：副本集的名称（"s0"）
2.成员：[]（members 是一个 mongod 实例的数组。暂时保持为空，我们将通过 push 命令添加成员。）

3.b 将 mongod 实例推入（添加）到配置对象中的 members 数组。在 mongo shell 中输入

```
config.members.push({"_id":0,"host":"localhost:37017"});
config.members.push({"_id":1,"host":"localhost:37018"});
config.members.push({"_id":2,"host":"localhost:37019"});
```

我们为每个 mongod 实例分配一个 _id 和一个 host。_id 可以是任何唯一数字，host 应该是运行该实例的服务器主机名，后跟端口号。

4.在 mongo shell 中通过以下命令初始化配置对象。

```
rs.initiate(config)
```

5.等待几秒钟，我们就有一个由 3 个 mongod 实例组成的副本集在服务器上运行。输入以下命令检查副本集状态，并识别哪个是主节点，哪个是从节点。

```
rs.status();
```

第 17.2 节：检查 MongoDB 副本集状态

使用以下命令检查副本集状态。

命令 ： rs.status()

连接任意一个副本成员并执行此命令，它将返回副本集的完整状态

示例 ：

```
   {
    "set" : "ReplicaName",
    "date" : ISODate("2016-09-26T07:36:04.935Z"),
    "myState" : 1,
    "term" : NumberLong(-1),
    "heartbeatIntervalMillis" : NumberLong(2000),
    "members" : [
            {
                    "_id" : 0,
                    "name" : "<IP>:<PORT>,
"health" : 1,
"state" : 1,
"stateStr" : "PRIMARY",
                    "uptime" : 5953744,
                    "optime" : Timestamp(1474875364, 36),
                    "optimeDate" : ISODate("2016-09-26T07:36:04Z"),
                    "electionTime" : Timestamp(1468921646, 1),
"electionDate" : ISODate("2016-07-19T09:47:26Z"),
                    "configVersion" : 6,
"self" : true
          },
            {
"_id" : 1,
                    "name" : "<IP>:<PORT>",
                    "health" : 1,
```


1. _id: the name of the replica Set ("s0")
2. members: [] (members is an array of mongod instances. lets keep this blank for now, we will add members via the push command.

3.b To Push(add) mongod instances to the members array in the config object. On the mongo shell type

```
config.members.push({"_id":0,"host":"localhost:37017"});
config.members.push({"_id":1,"host":"localhost:37018"});
config.members.push({"_id":2,"host":"localhost:37019"});
```

We assign each mongod instance an _id and an host. _id can be any unique number and the host should be the hostname of the server on which its running followed by the port number.

4. Initiate the config object by the following command in the mongo shell.

```
rs.initiate(config)
```

5. Give it a few seconds and we have a replica set of 3 mongod instances running on the server. type the following command to check the status of the replica set and to identify which one is primary and which one is secondary.

```
rs.status();
```

Section 17.2: Check MongoDB Replica Set states

Use the below command to check the replica set status.

Command : rs.status()

Connect any one of replica member and fire this command it will give the full state of the replica set

Example :

```
{
 "set" : "ReplicaName",
 "date" : ISODate("2016-09-26T07:36:04.935Z"),
 "myState" : 1,
 "term" : NumberLong(-1),
 "heartbeatIntervalMillis" : NumberLong(2000),
 "members" : [
        {
                "_id" : 0,
                "name" : "<IP>:<PORT>,
                "health" : 1,
                "state" : 1,
                "stateStr" : "PRIMARY",
                "uptime" : 5953744,
                "optime" : Timestamp(1474875364, 36),
                "optimeDate" : ISODate("2016-09-26T07:36:04Z"),
                "electionTime" : Timestamp(1468921646, 1),
                "electionDate" : ISODate("2016-07-19T09:47:26Z"),
                "configVersion" : 6,
                "self" : true
        },
        {
                "_id" : 1,
                "name" : "<IP>:<PORT>",
                "health" : 1,
```

```
"state" : 2,
                    "stateStr" : "SECONDARY",
                    "uptime" : 5953720,
"optime" : Timestamp(1474875364, 13),
                    "optimeDate" : ISODate("2016-09-26T07:36:04Z"),
                    "lastHeartbeat" : ISODate("2016-09-26T07:36:04.244Z"),
                    "lastHeartbeatRecv" : ISODate("2016-09-26T07:36:03.871Z"),
                    "pingMs" : NumberLong(0),
"syncingTo" : "10.9.52.55:10050",
                    "configVersion" : 6
            },
            {
"_id" : 2,
                    "name" : "<IP>:<PORT>",
                    "health" : 1,
"state" : 7,
"stateStr" : "ARBITER",
                    "uptime" : 5953696,
"lastHeartbeat" : ISODate("2016-09-26T07:36:03.183Z"),
                    "lastHeartbeatRecv" : ISODate("2016-09-26T07:36:03.715Z"),
                    "pingMs" : NumberLong(0),
"configVersion" : 6
          },
            {
"_id" : 3,
                    "name" : "<IP>:<PORT>",
                    "health" : 1,
"state" : 2,
                    "stateStr" : "SECONDARY",
                    "uptime" : 1984305,
"optime" : Timestamp(1474875361, 16),
                    "optimeDate" : ISODate("2016-09-26T07:36:01Z"),
                    "lastHeartbeat" : ISODate("2016-09-26T07:36:02.921Z"),
                    "lastHeartbeatRecv" : ISODate("2016-09-26T07:36:03.793Z"),
                    "pingMs" : NumberLong(22),
"lastHeartbeatMessage" : "syncing from: 10.9.52.56:10050",
                    "syncingTo" : "10.9.52.56:10050",
"configVersion" : 6
          }
    ],
"ok" : 1
   }
```

从以上内容我们可以了解到整个副本集的状态


```
                    "state" : 2,
                    "stateStr" : "SECONDARY",
                    "uptime" : 5953720,
                    "optime" : Timestamp(1474875364, 13),
                    "optimeDate" : ISODate("2016-09-26T07:36:04Z"),
                    "lastHeartbeat" : ISODate("2016-09-26T07:36:04.244Z"),
                    "lastHeartbeatRecv" : ISODate("2016-09-26T07:36:03.871Z"),
                    "pingMs" : NumberLong(0),
                    "syncingTo" : "10.9.52.55:10050",
                    "configVersion" : 6
            },
            {
                    "_id" : 2,
                    "name" : "<IP>:<PORT>",
                    "health" : 1,
                    "state" : 7,
                    "stateStr" : "ARBITER",
                    "uptime" : 5953696,
                    "lastHeartbeat" : ISODate("2016-09-26T07:36:03.183Z"),
                    "lastHeartbeatRecv" : ISODate("2016-09-26T07:36:03.715Z"),
                    "pingMs" : NumberLong(0),
                    "configVersion" : 6
            },
            {
                    "_id" : 3,
                    "name" : "<IP>:<PORT>",
                    "health" : 1,
                    "state" : 2,
                    "stateStr" : "SECONDARY",
                    "uptime" : 1984305,
                    "optime" : Timestamp(1474875361, 16),
                    "optimeDate" : ISODate("2016-09-26T07:36:01Z"),
                    "lastHeartbeat" : ISODate("2016-09-26T07:36:02.921Z"),
                    "lastHeartbeatRecv" : ISODate("2016-09-26T07:36:03.793Z"),
                    "pingMs" : NumberLong(22),
                    "lastHeartbeatMessage" : "syncing from: 10.9.52.56:10050",
                    "syncingTo" : "10.9.52.56:10050",
                    "configVersion" : 6
            }
    ],
    "ok" : 1
    }
```

From the above we can know the entire replica set status

第18章：MongoDB - 配置副本集以支持TLS/SSL

如何配置ReplicaSet以支持TLS/SSL？

我们将在本地环境中部署一个3节点的ReplicaSet，并使用自签名证书。请勿在生产环境中使用自签名证书。

如何将客户端连接到此ReplicaSet？

我们将连接一个Mongo Shell。

关于TLS/SSL、公钥基础设施（PKI）证书和证书颁发机构的描述超出本文件的范围。

第18.1节：如何配置ReplicaSet以支持TLS/SSL？

创建根证书

根证书（又称CA文件）将用于签署和识别您的证书。要生成它，请运行以下命令。

```
openssl req -nodes -out ca.pem -new -x509 -keyout ca.key
```

请妥善保管根证书及其密钥，二者都将用于签署您的证书。根证书也可能被您的客户端使用。

生成证书请求和私钥

生成证书签名请求（即 CSR）时，在"通用名称"（即 CN）字段中输入节点的准确主机名（或 IP）。其他字段必须具有完全相同的值。你可能需要修改你的/etc/hosts文件。

以下命令将生成 CSR 文件和 RSA 私钥（4096 位）。

```
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_node_1.key -out mongodb_node_1.csr
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_node_2.key -out mongodb_node_2.csr
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_node_3.key -out mongodb_node_3.csr
```

你必须为副本集的每个节点生成一个 CSR。请记住，每个节点的通用名称不同。不要基于同一个私钥生成多个 CSR。

你现在应该有 3 个 CSR 和 3 个私钥。

```
mongodb_node_1.key - mongodb_node_2.key - mongodb_node_3.key
mongodb_node_1.csr - mongodb_node_2.csr - mongodb_node_3.csr
```

签署你的证书请求

使用之前生成的 CA 文件（ca.pem）及其私钥（ca.key）通过运行以下命令为每个证书请求签名。

Chapter 18: MongoDB - Configure a ReplicaSet to support TLS/SSL

How to configure a ReplicaSet to support TLS/SSL?

We will deploy a 3 Nodes ReplicaSet in your local environment and we will use a self-signed certificate. Do not use a self-signed certificate in PRODUCTION.

How to connect your Client to this ReplicaSet?

We will connect a Mongo Shell.

A description of TLS/SSL, PKI (Public Key Infrastructure) certificates, and Certificate Authority is beyond the scope of this documentation.

Section 18.1: How to configure a ReplicaSet to support TLS/SSL?

Create the Root Certificate

The Root Certificate (aka CA File) will be used to sign and identify your certificate. To generate it, run the command below.

```
openssl req -nodes -out ca.pem -new -x509 -keyout ca.key
```

Keep the root certificate and its key carefully, both will be used to sign your certificates. The root certificate might be used by your client as well.

Generate the Certificate Requests and the Private Keys

When generating the Certificate Signing Request (aka CSR), **input the exact hostname (or IP) of your node in the Common Name (aka CN) field. The others fields must have exactly the same value.** You might need to modify your */etc/hosts* file.

The commands below will generate the CSR files and the RSA Private Keys (4096 bits).

```
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_node_1.key -out mongodb_node_1.csr
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_node_2.key -out mongodb_node_2.csr
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_node_3.key -out mongodb_node_3.csr
```

You must generate one CSR for each node of your ReplicaSet. Remember that the Common Name is not the same from one node to another. Don't base multiple CSRs on the same Private Key.

You must now have 3 CSRs and 3 Private Keys.

```
mongodb_node_1.key - mongodb_node_2.key - mongodb_node_3.key
mongodb_node_1.csr - mongodb_node_2.csr - mongodb_node_3.csr
```

Sign your Certificate Requests

Use the CA File (ca.pem) and its Private Key (ca.key) generated previously to sign each Certificate Request by running the commands below.

```
openssl x509 -req -in mongodb_node_1.csr -CA ca.pem -CAkey ca.key -set_serial 00 -out
mongodb_node_1.crt
openssl x509 -req -in mongodb_node_2.csr -CA ca.pem -CAkey ca.key -set_serial 00 -out
mongodb_node_2.crt
openssl x509 -req -in mongodb_node_3.csr -CA ca.pem -CAkey ca.key -set_serial 00 -out
mongodb_node_3.crt
```

你必须签署每个 CSR。

你现在应该有 3 个 CSR、3 个私钥和 3 个自签名证书。MongoDB 只会使用私钥和证书。

```
mongodb_node_1.key - mongodb_node_2.key - mongodb_node_3.key
mongodb_node_1.csr - mongodb_node_2.csr - mongodb_node_3.csr
mongodb_node_1.crt - mongodb_node_2.crt - mongodb_node_3.crt
```

每个证书对应一个节点。请仔细记住你为每个 CSR 提供的 CN / 主机名。

将每个节点的证书与其密钥合并

运行以下命令，将每个节点的证书与其密钥合并到一个文件中（MongoDB 要求）。

```
cat mongodb_node_1.key mongodb_node_1.crt > mongodb_node_1.pem
cat mongodb_node_2.key mongodb_node_2.crt > mongodb_node_2.pem
cat mongodb_node_3.key mongodb_node_3.crt > mongodb_node_3.pem
```

你现在应该有 3 个 PEM 文件。

```
mongodb_node_1.pem - mongodb_node_2.pem - mongodb_node_3.pem
```

部署您的副本集

我们假设您的pem文件位于当前文件夹以及data/data1、data/data2和
data/data3中。

运行以下命令以部署监听端口为27017、27018和27019的3节点副本集。

```
mongod --dbpath data/data_1 --replSet rs0 --port 27017 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_1.pem
mongod --dbpath data/data_2 --replSet rs0 --port 27018 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_2.pem
mongod --dbpath data/data_3 --replSet rs0 --port 27019 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_3.pem
```

您现在已经在本地环境部署了一个3节点副本集，且所有事务均已加密。
未使用TLS，您无法连接到此副本集。

为双向SSL/双向信任部署您的副本集

要强制客户端提供客户端证书（双向SSL），您必须在运行实例时添加CA文件。

```
mongod --dbpath data/data_1 --replSet rs0 --port 27017 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_1.pem --sslCAFile ca.pem
mongod --dbpath data/data_2 --replSet rs0 --port 27018 --sslMode requireSSL --sslPEMKeyFile
```


openssl x509 -req -in mongodb_node_1.csr -CA ca.pem -CAkey ca.key -set_serial 00 -out
mongodb_node_1.crt
openssl x509 -req -in mongodb_node_2.csr -CA ca.pem -CAkey ca.key -set_serial 00 -out
mongodb_node_2.crt
openssl x509 -req -in mongodb_node_3.csr -CA ca.pem -CAkey ca.key -set_serial 00 -out
mongodb_node_3.crt

You must sign each CSR.

Your must now have 3 CSRs, 3 Private Keys and 3 self-signed Certificates. Only the Private Keys and the Certificates will be used by MongoDB.

mongodb_node_1.key - mongodb_node_2.key - mongodb_node_3.key
mongodb_node_1.csr - mongodb_node_2.csr - mongodb_node_3.csr
mongodb_node_1.crt - mongodb_node_2.crt - mongodb_node_3.crt

Each certificate corresponds to one node. Remember carefully which CN / hostname your gave to each CSR.

Concat each Node Certificate with its key

Run the commands below to concat each Node Certificate with its key in one file (MongoDB requirement).

cat mongodb_node_1.key mongodb_node_1.crt > mongodb_node_1.pem
cat mongodb_node_2.key mongodb_node_2.crt > mongodb_node_2.pem
cat mongodb_node_3.key mongodb_node_3.crt > mongodb_node_3.pem

Your must now have 3 PEM files.

mongodb_node_1.pem - mongodb_node_2.pem - mongodb_node_3.pem

Deploy your ReplicaSet

We will assume that your pem files are located in your current folder as well as data/data1, data/data2 and data/data3.

Run the commands below to deploy your 3 Nodes ReplicaSet listening on port 27017, 27018 and 27019.

mongod --dbpath data/data_1 --replSet rs0 --port 27017 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_1.pem
mongod --dbpath data/data_2 --replSet rs0 --port 27018 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_2.pem
mongod --dbpath data/data_3 --replSet rs0 --port 27019 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_3.pem

You now have a 3 Nodes ReplicaSet deployed on your local environment and all their transactions are encrypted.
You cannot connect to this ReplicaSet without using TLS.

Deploy your ReplicaSet for Mutual SSL / Mutual Trust

To force your client to provide a Client Certificate (Mutual SSL), you must add the CA File when running your instances.

mongod --dbpath data/data_1 --replSet rs0 --port 27017 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_1.pem --sslCAFile ca.pem
mongod --dbpath data/data_2 --replSet rs0 --port 27018 --sslMode requireSSL --sslPEMKeyFile

```
mongodb_node_2.pem --sslCAFile ca.pem
mongod --dbpath data/data_3 --replSet rs0 --port 27019 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_3.pem --sslCAFile ca.pem
```

您现在已经在本地环境部署了一个3节点副本集，且所有事务均已加密。

如果不使用 TLS 或不提供由您的 CA 信任的客户端证书，您无法连接到此副本集。

第18.2节：如何将客户端（Mongo Shell）连接到副本集？

无双向 SSL

在此示例中，我们可能会使用您在"*如何配置副本集以支持 TLS/SSL？*"部分生成的 CA 文件（ca.pem）。我们假设 CA 文件位于您当前的文件夹中。

我们假设您的 3 个节点运行在 mongo1:27017、mongo2:27018 和 mongo3:27019。（您可能需要修改您的 /etc/hosts 文件。）

从 MongoDB 3.2.6 开始，如果您的 CA 文件已注册到操作系统的信任存储中，您可以在不提供 CA 文件的情况下连接到您的副本集。

```
mongo --ssl --主机 rs0/mongo1:27017,mongo2:27018,mongo3:27019
```

否则您必须提供 CA 文件。

```
mongo --ssl --sslCAFile ca.pem --主机 rs0/mongo1:27017,mongo2:27018,mongo3:27019
```

您现在已连接到您的副本集，Mongo Shell 与副本集之间的所有事务均已加密。

使用双向 SSL

如果您的副本集要求客户端证书，您必须提供由副本集部署所用 CA 签发的证书。生成客户端证书的步骤与生成服务器证书的步骤几乎相同。

实际上，您只需在 CSR 创建过程中修改通用名称字段。通用名称字段中不再只填写一个节点主机名，而是需要填写所有副本集主机名，主机名之间用逗号分隔。

```
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_client.key -out mongodb_client.csr
...
通用名称(e.g. 服务器 FQDN 或您的名字) []: mongo1,mongo2,mongo3
```

如果通用名称字段过长（超过 64 字节），您可能会遇到通用名称大小限制的问题。
要绕过此限制，生成证书签名请求（CSR）时必须使用 SubjectAltName。

```
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_client.key -out mongodb_client.csr -
config <(
cat <<-EOF
[req]
default_bits = 4096
prompt = no
default_md = sha256
req_extensions = req_ext
```



```
mongodb_node_2.pem --sslCAFile ca.pem
mongod --dbpath data/data_3 --replSet rs0 --port 27019 --sslMode requireSSL --sslPEMKeyFile
mongodb_node_3.pem --sslCAFile ca.pem
```

You now have a 3 Nodes ReplicaSet deployed on your local environment and all their transactions are encrypted. You cannot connect to this ReplicaSet without using TLS or without providing a Client Certificate trusted by your CA.

Section 18.2: How to connect your Client (Mongo Shell) to a ReplicaSet?

No Mutual SSL

In this example, we might use the CA File (ca.pem) that you generated during the "*How to configure a ReplicaSet to support TLS/SSL?*" section. We will assume that the CA file is located in your current folder.

We will assume that your 3 nodes are running on mongo1:27017, mongo2:27018 and mongo3:27019. (You might need to modify your */etc/hosts* file.)

From MongoDB 3.2.6, if your CA File is registered in your Operating System Trust Store, you can connect to your ReplicaSet without providing the CA File.

```
mongo --ssl --host rs0/mongo1:27017,mongo2:27018,mongo3:27019
```

Otherwise you must provide the CA File.

```
mongo --ssl --sslCAFile ca.pem --host rs0/mongo1:27017,mongo2:27018,mongo3:27019
```

You are now connected to your ReplicaSet and all the transactions between your Mongo Shell and your ReplicaSet are encrypted.

With Mutual SSL

If your ReplicaSet asks for a Client Certificate, you must provide one signed by the CA used by the ReplicaSet Deployment. The steps to generate the Client Certificate are almost the same as the ones to generate the Server Certificate.

Indeed, you just need to modify the Common Name Field during the CSR creation. Instead of providing 1 Node Hostname in the Common Name Field, **you need to provide all the ReplicaSet Hostnames separated by a comma**.

```
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_client.key -out mongodb_client.csr
...
Common Name (e.g. server FQDN or YOUR name) []: mongo1,mongo2,mongo3
```

You might face the Common Name size limitation if the Common Name field is too long (more than 64 bytes long). To bypass this limitation, you must use the SubjectAltName when generating the CSR.

```
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongodb_client.key -out mongodb_client.csr -
config <(
cat <<-EOF
[req]
default_bits = 4096
prompt = no
default_md = sha256
req_extensions = req_ext
```

```
distinguished_name = dn

[ dn ]
CN = .

[ req_ext ]
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = mongo1
DNS.2 = mongo2
DNS.3 = mongo3
EOF
)
```

然后你使用CA证书和密钥对CSR进行签名。

```
openssl x509 -req -in mongodb_client.csr  -CA ca.pem -CAkey ca.key -set_serial 00 -out
mongodb_client.crt
```

最后，你将密钥和签名证书合并。

```
cat mongodb_client.key mongodb_client.crt > mongodb_client.pem
```

要连接到你的副本集，现在可以提供新生成的客户端证书。

```
mongo --ssl --sslCAFile ca.pem --host rs0/mongo1:27017,mongo2:27018,mongo3:27019 --sslPEMKeyFile
mongodb_client.pem
```

您现在已连接到您的副本集，Mongo Shell 与副本集之间的所有事务均已加密。


```
distinguished_name = dn

[ dn ]
CN = .

[ req_ext ]
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = mongo1
DNS.2 = mongo2
DNS.3 = mongo3
EOF
)
```

Then you sign the CSR using the CA certificate and key.

```
openssl x509 -req -in mongodb_client.csr  -CA ca.pem -CAkey ca.key -set_serial 00 -out
mongodb_client.crt
```

Finally, you concat the key and the signed certificate.

```
cat mongodb_client.key mongodb_client.crt > mongodb_client.pem
```

To connect to your ReplicaSet, you can now provide the newly generated Client Certificate.

```
mongo --ssl --sslCAFile ca.pem --host rs0/mongo1:27017,mongo2:27018,mongo3:27019 --sslPEMKeyFile
mongodb_client.pem
```

You are now connected to your ReplicaSet and all the transactions between your Mongo Shell and your ReplicaSet are encrypted.

第19章：MongoDB中的认证机制

认证是验证客户端身份的过程。当启用访问控制，即授权时，
MongoDB要求所有客户端进行身份验证，以确定其访问权限。

MongoDB支持多种认证机制，客户端可以使用这些机制来验证其身份。这些机制允许MongoDB集成到你现有的认证系统中。

第19.1节：认证机制

MongoDB 支持多种身份验证机制。

客户端和用户认证机制

- SCRAM-SHA-1
- X.509 证书认证
- MongoDB 挑战响应认证（MONGODB-CR）
- LDAP 代理认证，以及
- Kerberos 认证

内部认证机制

- 密钥文件
- X.509

Chapter 19: Authentication Mechanisms in MongoDB

Authentication is the process of verifying the identity of a client. When access control, i.e. authorization, is enabled, MongoDB requires all clients to authenticate themselves in order to determine their access.

MongoDB supports a number of authentication mechanisms that clients can use to verify their identity. These mechanisms allow MongoDB to integrate into your existing authentication system.

Section 19.1: Authentication Mechanisms

MongoDB supports multiple authentication mechanisms.

Client and User Authentication Mechanisms

- SCRAM-SHA-1
- X.509 Certificate Authentication
- MongoDB Challenge and Response (MONGODB-CR)
- LDAP proxy authentication, and
- Kerberos authentication

Internal Authentication Mechanisms

- Keyfile
- X.509

第20章：MongoDB 授权模型

授权基本上是验证用户权限。MongoDB 支持不同类型的授权模型。1.
基于角色的访问控制
 角色是一组权限，是对资源的操作。这些权限授予用户在特定命名空间（数据库）上的访问权。操作是在资源上执行的。资源是数据库中任何持有状态的对象。

第20.1节：内置角色

内置数据库用户角色和数据库管理角色存在于每个数据库中。

数据库用户角色

1.只读
2.读写

Chapter 20: MongoDB Authorization Model

Authorization is the basically verifies user privileges. MongoDB support different kind of authorization models. 1. **Role base access control**
 Role are group of privileges, actions over resources. That are gain to users over a given namespace (Database). Actions are performs on resources. Resources are any object that hold state in database.

Section 20.1: Build-in Roles

Built-in database user roles and database administration roles roles exist in each database.

Database User Roles

1. `read`
2. `readwrite`

第21章：配置

参数	默认
systemLog.verbosity	0
systemLog.quiet	false
systemLog.traceAllExceptions	false
systemLog.syslogFacility	用户
systemLog.path	-
systemLog.logAppend	false
systemLog.logRotate	重命名
systemLog.destination	标准输出
systemLog.timeStampFormat	iso8601-local
systemLog.component.accessControl.verbosity	0
systemLog.component.command.verbosity	0
systemLog.component.control.verbosity	0
systemLog.component.ftdc.verbosity	0
systemLog.component.geo.verbosity	0
systemLog.component.index.verbosity	0
systemLog.component.network.verbo	0
systemLog.component.query.verbosity	0
systemLog.component.replication.verbosity	0
systemLog.component.sharding.verbosity	0
systemLog.component.storage.verbosity	0
systemLog.component.storage.journal.verbosity	0
systemLog.component.write.verbosity	0
processManagement.fork	false
processManagement.pidFilePath	无
网络端口	27017
网络绑定IP	0.0.0.0
网络最大传入连接数	65536
网络对象校验	true
网络IPv6	false
网络Unix域套接字启用	true
net.unixDomainSocket.pathPrefix	/tmp
net.unixDomainSocket.filePermissions	0700
net.http.enabled	false
net.http.JSONPEnabled	false
net.http.RESTInterfaceEnabled	false
net.ssl.sslOnNormalPorts	false
net.ssl.mode	disabled
net.ssl.PEMKeyFile	无
net.ssl.PEMKeyPassword	无
net.ssl.clusterFile	无
net.ssl.clusterPassword	无
net.ssl.CAFile	无

Chapter 21: Configuration

Parameter	Default
systemLog.verbosity	0
systemLog.quiet	false
systemLog.traceAllExceptions	false
systemLog.syslogFacility	user
systemLog.path	-
systemLog.logAppend	false
systemLog.logRotate	rename
systemLog.destination	stdout
systemLog.timeStampFormat	iso8601-local
systemLog.component.accessControl.verbosity	0
systemLog.component.command.verbosity	0
systemLog.component.control.verbosity	0
systemLog.component.ftdc.verbosity	0
systemLog.component.geo.verbosity	0
systemLog.component.index.verbosity	0
systemLog.component.network.verbo	0
systemLog.component.query.verbosity	0
systemLog.component.replication.verbosity	0
systemLog.component.sharding.verbosity	0
systemLog.component.storage.verbosity	0
systemLog.component.storage.journal.verbosity	0
systemLog.component.write.verbosity	0
processManagement.fork	false
processManagement.pidFilePath	none
net.port	27017
net.bindIp	0.0.0.0
net.maxIncomingConnections	65536
net.wireObjectCheck	true
net.ipv6	false
net.unixDomainSocket.enabled	true
net.unixDomainSocket.pathPrefix	/tmp
net.unixDomainSocket.filePermissions	0700
net.http.enabled	false
net.http.JSONPEnabled	false
net.http.RESTInterfaceEnabled	false
net.ssl.sslOnNormalPorts	false
net.ssl.mode	disabled
net.ssl.PEMKeyFile	none
net.ssl.PEMKeyPassword	none
net.ssl.clusterFile	none
net.ssl.clusterPassword	none
net.ssl.CAFile	none

net.ssl.CRLFile	无
net.ssl.allowConnectionsWithoutCertificates	false
net.ssl.allowInvalidCertificates	false
net.ssl.allowInvalidHostnames	false
net.ssl.disabledProtocols	无
net.ssl.FIPSMode	false

第21.1节：使用特定配置文件启动mongo

使用--config参数。

```
$ /bin/mongod --config /etc/mongod.conf
$ /bin/mongos --config /etc/mongos.conf
```

注意-f是--config的简写同义词。

net.ssl.CRLFile	none
net.ssl.allowConnectionsWithoutCertificates	false
net.ssl.allowInvalidCertificates	false
net.ssl.allowInvalidHostnames	false
net.ssl.disabledProtocols	none
net.ssl.FIPSMode	false

Section 21.1: Starting mongo with a specific config file

Using the `--config` flag.

```
$ /bin/mongod --config /etc/mongod.conf
$ /bin/mongos --config /etc/mongos.conf
```

Note that `-f` is the shorter synonym for `--config`.

第22章：备份和恢复数据

第22.1节：本地默认mongod实例的基本mongodump

```
mongodump --db mydb --gzip --out "mydb.dump.$(date +%F_%R)"
```

该命令将把本地mongod的'mydb'数据库导出为bson格式的gzip压缩档案，存储到'mydb.dump.{时间戳}'目录中

第22.2节：本地默认mongod备份的基本mongorestore

```
mongorestore --db mydb mydb.dump.2016-08-27_12:44/mydb --drop --gzip
```

此命令将首先删除您当前的“mydb”数据库，然后从“mydb mydb.dump.2016-08-27_12:44/mydb”归档转储文件中恢复您的 gzip 压缩的 bson 转储。

第22.3节：使用mongoimport导入JSON

示例邮政编码数据集zipcodes.json存储在c:\Users\yc03ak1\Desktop\zips.json

```
{ "_id" : "01001", "city" : "阿加姆", "loc" : [ -72.622739, 42.070206 ], "pop" : 15338, "state" :
"马萨诸塞州" }
{ "_id" : "01002", "city" : "库什曼", "loc" : [ -72.51564999999999, 42.377017 ], "pop" : 36963,
"state" : "马萨诸塞州" }
{ "_id" : "01005", "city" : "巴雷", "loc" : [ -72.10835400000001, 42.409698 ], "pop" : 4546,
"state" : "马萨诸塞州" }
{ "_id" : "01007", "city" : "贝尔彻镇", "loc" : [ -72.41095300000001, 42.275103 ], "pop" :
10579, "state" : "马萨诸塞州" }
{ "_id" : "01008", "city" : "布兰福德", "loc" : [ -72.936114, 42.182949 ], "pop" : 1240, "state" :
"马萨诸塞州" }
{ "_id" : "01010", "city" : "布里姆菲尔德", "loc" : [ -72.188455, 42.116543 ], "pop" : 3706, "state" :
"马萨诸塞州" }
{ "_id" : "01011", "city" : "切斯特", "loc" : [ -72.988761, 42.279421 ], "pop" : 1688, "state" :
"马萨诸塞州" }
```

将此数据集导入名为“test”的数据库和名为“zips”的集合中

```
C:\Users\yc03ak1>mongoimport --db test --collection "zips" --drop --type json --host
"localhost:47019"  --file "c:\Users\yc03ak1\Desktop\zips.json"
```

- --db ：要导入数据的数据库名称
- --collection：数据库中要导入数据的集合名称
- --drop ：导入前先删除集合
- --type ：需要导入的文档类型，默认是 JSON
- --host ：MongoDB 的主机和端口，数据将导入到此处
- --file ：json 文件所在路径

输出：

```
2016-08-10T20:10:50.159-0700    已连接到：localhost:47019
```


Chapter 22: Backing up and Restoring Data

Section 22.1: Basic mongodump of local default mongod instance

```
mongodump --db mydb --gzip --out "mydb.dump.$(date +%F_%R)"
```

This command will dump a bson gzipped archive of your local mongod 'mydb' database to the 'mydb.dump.{timestamp}' directory

Section 22.2: Basic mongorestore of local default mongod dump

```
mongorestore --db mydb mydb.dump.2016-08-27_12:44/mydb --drop --gzip
```

This command will first drop your current 'mydb' database and then restore your gzipped bson dump from the 'mydb mydb.dump.2016-08-27_12:44/mydb' archive dump file.

Section 22.3: mongoimport with JSON

Sample zipcode dataset in zipcodes.json stored in c:\Users\yc03ak1\Desktop\zips.json

```
{ "_id" : "01001", "city" : "AGAWAM", "loc" : [ -72.622739, 42.070206 ], "pop" : 15338, "state" :
"MA" }
{ "_id" : "01002", "city" : "CUSHMAN", "loc" : [ -72.51564999999999, 42.377017 ], "pop" : 36963,
"state" : "MA" }
{ "_id" : "01005", "city" : "BARRE", "loc" : [ -72.10835400000001, 42.409698 ], "pop" : 4546,
"state" : "MA" }
{ "_id" : "01007", "city" : "BELCHERTOWN", "loc" : [ -72.41095300000001, 42.275103 ], "pop" :
10579, "state" : "MA" }
{ "_id" : "01008", "city" : "BLANDFORD", "loc" : [ -72.936114, 42.182949 ], "pop" : 1240, "state" :
"MA" }
{ "_id" : "01010", "city" : "BRIMFIELD", "loc" : [ -72.188455, 42.116543 ], "pop" : 3706, "state" :
"MA" }
{ "_id" : "01011", "city" : "CHESTER", "loc" : [ -72.988761, 42.279421 ], "pop" : 1688, "state" :
"MA" }
```

to import this data-set to the database named "test" and collection named "zips"

```
C:\Users\yc03ak1>mongoimport --db test --collection "zips" --drop --type json --host
"localhost:47019"  --file "c:\Users\yc03ak1\Desktop\zips.json"
```

- --db : name of the database where data is to be imported to
- --collection: name of the collection in the database where data is to be improted
- --drop : drops the collection first before importing
- --type : document type which needs to be imported. default JSON
- --host : mongodb host and port on which data is to be imported.
- --file : path where the json file is

output :

```
2016-08-10T20:10:50.159-0700    connected to: localhost:47019
```

```
2016-08-10T20:10:50.163-0700    正在删除：test.zips
2016-08-10T20:10:53.155-0700    [################........] test.zips    2.1 MB/3.0 MB (68.5%)
2016-08-10T20:10:56.150-0700    [########################] test.zips    3.0 MB/3.0 MB (100.0%)
2016-08-10T20:10:57.819-0700    [########################] test.zips    3.0 MB/3.0 MB (100.0%)
2016-08-10T20:10:57.821-0700    导入了 29353 个文档
```

第22.4节：使用mongoimport导入CSV

示例测试数据集CSV文件存储在位置 c:\Users\yc03ak1\Desktopesting.csv

```
_id    城市    位置    人口    州
1    A    [10.0, 20.0]    2222    PQE
2    B    [10.1, 20.1]    22122    RW
3    C    [10.2, 20.0]    255222    RWE
4    D    [10.3, 20.3]    226622    SFDS
5    E    [10.4, 20.0]    222122    FDS
```

将此数据集导入名为“test”的数据库和名为“sample”的集合中

```
C:\Users\yc03ak1>mongoimport --db test --collection "sample" --drop --type csv --headerline --host
"localhost:47019"  --file "c:\Users\yc03ak1\Desktopesting.csv"
```

- --headerline ：使用CSV文件的第一行作为JSON文档的字段

输出：

```
2016-08-10T20:25:48.572-0700    已连接到：localhost:47019
2016-08-10T20:25:48.576-0700    正在删除：test.sample
2016-08-10T20:25:49.109-0700    导入了 5 个文档
```

或者

```
C:\Users\yc03ak1>mongoimport --db test --collection "sample" --drop --type csv --fields
_id,city,loc,pop,state --host "localhost:47019"  --file "c:\Users\yc03ak1\Desktopesting.csv"
```

- --fields ：以逗号分隔的字段列表，需要导入到 JSON 文档中。输出：

```
2016-08-10T20:26:48.978-0700    已连接到：localhost:47019
2016-08-10T20:26:48.982-0700    正在删除：test.sample
2016-08-10T20:26:49.611-0700    已导入 6 个文档
```



```
2016-08-10T20:10:50.163-0700    dropping: test.zips
2016-08-10T20:10:53.155-0700    [################........] test.zips    2.1 MB/3.0 MB (68.5%)
2016-08-10T20:10:56.150-0700    [########################] test.zips    3.0 MB/3.0 MB (100.0%)
2016-08-10T20:10:57.819-0700    [########################] test.zips    3.0 MB/3.0 MB (100.0%)
2016-08-10T20:10:57.821-0700    imported 29353 documents
```

Section 22.4: mongoimport with CSV

Sample test dataset CSV file stored at the location c:\Users\yc03ak1\Desktop\testing.csv

```
_id    city    loc    pop    state
1    A    [10.0, 20.0]    2222    PQE
2    B    [10.1, 20.1]    22122    RW
3    C    [10.2, 20.0]    255222    RWE
4    D    [10.3, 20.3]    226622    SFDS
5    E    [10.4, 20.0]    222122    FDS
```

to import this data-set to the database named "test" and collection named "sample"

```
C:\Users\yc03ak1>mongoimport --db test --collection "sample" --drop --type csv --headerline --host
"localhost:47019"  --file "c:\Users\yc03ak1\Desktop\testing.csv"
```

- --headerline : use the first line of the csv file as the fields for the json document

output :

```
2016-08-10T20:25:48.572-0700    connected to: localhost:47019
2016-08-10T20:25:48.576-0700    dropping: test.sample
2016-08-10T20:25:49.109-0700    imported 5 documents
```

OR

```
C:\Users\yc03ak1>mongoimport --db test --collection "sample" --drop --type csv --fields
_id,city,loc,pop,state --host "localhost:47019"  --file "c:\Users\yc03ak1\Desktop\testing.csv"
```

- --fields : comma separated list of fields which needs to be imported in the json document. Output:

```
2016-08-10T20:26:48.978-0700    connected to: localhost:47019
2016-08-10T20:26:48.982-0700    dropping: test.sample
2016-08-10T20:26:49.611-0700    imported 6 documents
```

第23章：升级 MongoDB 版本

如何在不同平台和版本的机器上更新 MongoDB 版本。

第23.1节：使用 apt 在 Ubuntu 16.04 上升级到 3.4

必须先有 3.2 版本才能升级到 3.4。本示例假设您正在使用apt。

1. sudo service mongod stop
2. sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 0C49F3730359A14518585931BC711F9BA15703C6
3. echo "deb [arch=amd64,arm64] http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list
4. sudo apt-get update
5. sudo apt-get upgrade
6. sudo service mongod start

确保新版本正在运行，使用 mongo。Shell 会打印出 MongoDB 服务器版本，应该是 3.4 版本。

Chapter 23: Upgrading MongoDB version

How to update the version of MongoDB on your machine on different platforms and versions.

Section 23.1: Upgrading to 3.4 on Ubuntu 16.04 using apt

You must have 3.2 to be able to upgrade to 3.4. This example assumes you are using apt.

1. sudo service mongod stop
2. sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 0C49F3730359A14518585931BC711F9BA15703C6
3. echo "deb [arch=amd64,arm64] http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list
4. sudo apt-get update
5. sudo apt-get upgrade
6. sudo service mongod start

Ensure the new version is running with mongo. The shell will print out the MongoDB server version that should be 3.4 now.

鸣谢

非常感谢所有来自Stack Overflow Documentation的人员提供此内容，
更多更改可发送至web@petercv.com以发布或更新新内容

阿卜杜勒·雷赫曼·赛义德	第1章
ADIMO	第16章
Antti_M	第23章
Ashari	第1章
Avindu Hewa	第四章
bappr	第18章
Batsu	第9章
chridam	第10章
康斯坦丁·盖伊	第9章和第12章
德林	第14章
dev ツ	第13章
埃米尔·布尔佐	第13章
伊纳穆尔·哈桑	第1章和第8章
fracz	第2章和第3章
grape	第8章
gypsyCoder	第11章
霍夫迈斯特	第8章
ipip	第1章
伊山·索尼	第二章
贾因	第二章
杰瑞	第二章
约翰尼HK	第二章
胡安·卡洛斯·法拉	第9章
凯鲁姆·塞纳纳耶克	第二章
KrisVos130	第二章
库汉	第6章
拉克马尔·维塔纳奇	第2章和第8章
洛伊克M	第8章
卢赞·巴拉尔	第19章
马尔科	第二章
马特·克拉克	第21章
尼克·科特雷尔	第9章
尼罗山·拉纳帕蒂	第19章和第20章
oggo	第四章
普罗森·戈什	第1章、第2章、第4章和第7章
RaR	第8章和第9章
雷努卡拉迪亚	第1章和第2章
罗特姆	第二章
肖恩·赖利	第1章
塞尔瓦·库马尔	第15章
sergiuz	第14章
施拉巴尼	第二章
索默工程	第四章
steveinatorx	第8章
styvane	第二章
托马斯·博尔曼斯	第二章
蒂姆	第12章

Credits

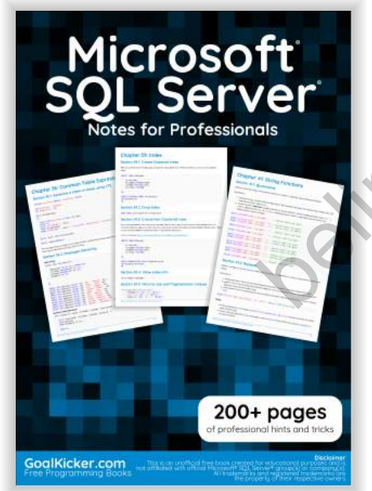
Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content,
more changes can be sent to web@petercv.com for new content to be published or updated

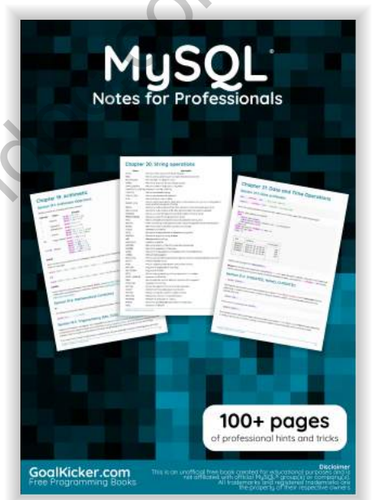
Abdul Rehman Sayed	Chapter 1
ADIMO	Chapter 16
Antti_M	Chapter 23
Ashari	Chapter 1
Avindu Hewa	Chapter 4
bappr	Chapter 18
Batsu	Chapter 9
chridam	Chapter 10
Constantin Guay	Chapters 9 and 12
Derlin	Chapter 14
dev ツ	Chapter 13
Emil Burzo	Chapter 13
Enamul Hassan	Chapters 1 and 8
fracz	Chapters 2 and 3
grape	Chapter 8
gypsyCoder	Chapter 11
HoefMeistert	Chapter 8
ipip	Chapter 1
Ishan Soni	Chapter 2
Jain	Chapter 2
jerry	Chapter 2
JohnnyHK	Chapter 2
Juan Carlos Farah	Chapter 9
Kelum Senanayake	Chapter 2
KrisVos130	Chapter 2
Kuhan	Chapter 6
Lakmal Vithanage	Chapters 2 and 8
LoicM	Chapter 8
Luzan Baral	Chapter 19
Marco	Chapter 2
Matt Clark	Chapter 21
Nic Cottrell	Chapter 9
Niroshan Ranapathi	Chapters 19 and 20
oggo	Chapter 4
Prosen Ghosh	Chapters 1, 2, 4 and 7
RaR	Chapters 8 and 9
Renukaradhya	Chapters 1 and 2
Rotem	Chapter 2
Sean Reilly	Chapter 1
Selva Kumar	Chapter 15
sergiuz	Chapter 14
Shrabanee	Chapter 2
SommerEngineering	Chapter 4
steveinatorx	Chapter 8
styvane	Chapter 2
Thomas Bormans	Chapter 2
tim	Chapter 12

titogeo	第1章、第8章和第9章
托马斯·卡尼巴诺	第2章和第9章
user641887	第17章和第22章
WAF	第1章
yellowB	第5章
扎农	第12章

titogeo	Chapters 1, 8 and 9
Tomás Cañibano	Chapters 2 and 9
user641887	Chapters 17 and 22
WAF	Chapter 1
yellowB	Chapter 5
Zanon	Chapter 12

你可能也喜欢





You may also like