# 安装和基本

1.安装MongoDB

- 安装

- 配置环境变量

C:\Program Files\MongoDB\Server\3.2\bin

- 在c盘根目录

- 创建一个文件夹 data

- 在data中创建一个文件夹db

- 打开cmd命令行窗口

- 输入 mongod 启动mongodb服务器

- 32位注意：

启动服务器时，需要输入如下内容

mongod --storageEngine=mmapv1

mongod --dbpath 数据库路径 --port 端口号

- 在打开一个cmd窗口

- 输入 mongo 连接mongodb ，出现 >

- 数据库（database）

- 数据库的服务器

- 服务器用来保存数据

- mongod 用来启动服务器

- 数据库的客户端

- 客户端用来操作服务器，对数据进行增删改查的操作

- mongo 用来启动客户端

- 将MongoDB设置为系统服务，可以自动在后台启动，不需要每次都手动启动

1.在c盘根目录创建data

- 在data下创建db和log文件夹

2.创建配置文件

在目录 C:\Program Files\MongoDB\Server\3.2 下添加一个配置文件

mongod.cfg

3.以管理员的身份打开命令行窗口

4.执行如下的命令

sc.exe create MongoDB binPath= "\"C:\Program Files\MongoDB\Server\3.2\bin\mongod.exe\" --service --config=\"C:\Program Files\MongoDB\Server\3.2\mongod.cfg\"" DisplayName= "MongoDB" start= "auto"

sc.exe create MongoDB binPath= "\"mongod的bin目录\mongod.exe\" --service --config=\"mongo的安装目录\mongod.cfg\"" DisplayName= "MongoDB" start= "auto"

5.启动mongodb服务

6.如果启动失败，证明上边的操作有误，

在控制台输入 sc delete MongoDB 删除之前配置的服务

然后从第一步再来一次

- 基本概念

数据库（database）

集合（collection）

文档（document）

- 在MongoDB中，数据库和集合都不需要手动创建，

当我们创建文档时，如果文档所在的集合或数据库不存在会自动创建数据库和集合

- 基本指令

show dbs

show databases

- 显示当前的所有数据库

use 数据库名

- 进入到指定的数据库中

db

- db表示的是当前所处的数据库

show collections

- 显示数据库中所有的集合

- 数据库的CRUD（增删改查）的操作

- 向数据库中插入文档

db.<collection>.insert(doc)

- 向集合中插入一个文档

- 例子：向test数据库中的，stus集合中插入一个新的学生对象

{name:"孙悟空",age:18,gender:"男"}

db.stus.insert({name:"孙悟空",age:18,gender:"男"})

db.<collection>.find()

- 查询当前集合中的所有的文档

MongoDB

- MongoDB是一个NoSQL的数据库

- MongoDB是一款文档型数据库

- 数据库指的就是一个存储数据的仓库

数据库可以使我们完成对数据的持久化的操作

- MongoDB数据库中存储的数据的基本单位就是文档，

MongoDB中存储的就是文档，所谓文档其实就是一个“JSON”

- MongoDB中的“JSON”我们称为BSON，比普通的JSON的功能要更加的强大

- MongoDB数据库使用的是JavaScript进行操作的，在MongoDB含有一个对ES标准实现的引擎，

在MongoDB中所有ES中的语法中都可以使用

- MongoDB的基本的指令

- 启动服务器

mongod --dbpath 路径 --port 端口号

- 启动客户端

mongo

- MongoDB的CRUD的操作

- 基本操作

use 数据库

- 进入指定的数据库

show dbs

- 显示所有的数据库

show collections

- 显示数据库中所有的集合

db

- 显示当前所在的数据库

- 向数据库中插入文档

- db.collection.insert()

- insert()可以向集合中插入一个或多个文档

- db.collection.insertOne()

- 向集合中插入一个文档

- db.collection.insertMany()

- 向集合中插入多个文档

- 查询数据库中的文档

- db.collection.find()

- 可以根据指定条件从集合中查询所有符合条件的文档

- 返回的是一个数组

- db.collection.findOne()

- 查询第一个符合条件的文档

- 返回的是一个对象

- db.collection.find().count()

- 查询符合条件的文档的数量

- 修改数据库中的文档

- db.collection.update()

- 可以修改、替换集合中的一个或多个文档

- db.collection.updateOne()

- 修改集合中的一个文档

- db.collection.updateMany()

- 修改集合中的多个文档

- db.collection.replaceOne()

- 替换集合中的一个文档

- 删除集合中的文档

- db.collection.remove()

- 删除集合中的一个或多个文档（默认删除多个）

- db.collection.deleteOne()

- 删除集合中的一个文档

- db.collection.deleteMany()

- 删除集合中的多个文档

- 清空一个集合

db.collection.remove({})

- 删除一个集合

db.collection.drop()

- 删除一个数据库

db.dropDatabase()

# 概述

MongoDB 是一款跨平台、面向文档的数据库。用它创建的数据库可以实现高性能、高可用性，并且能够轻松扩展。MongoDB 的运行方式主要基于两个概念：集合（collection）与文档（document）。

## 数据库

数据库是集合的实际容器。每一数据库都在文件系统中有自己的一组文件。一个 MongoDB 服务器通常有多个数据库。

## 集合

集合就是一组 MongoDB 文档。它相当于关系型数据库（RDBMS）中的表这种概念。集合位于单独的一个数据库中。集合不能执行模式（schema）。一个集合内的多个文档可以有多个不同的字段。一般来说，集合中的文档都有着相同或相关的目的。

## 文档

文档就是一组键-值对。文档有着动态的模式，这意味着同一集合内的文档不需要具有同样的字段或结构。

下表展示了关系型数据库与 MongoDB 在术语上的对比：

| **关系型数据库** | **MongoDB** |
| --- | --- |
| 数据库 | 数据库 |
| 表 | 集合 |
| 行 | 文档 |
| 列 | 字段 |
| 表 Join | 内嵌文档 |
| 主键 | 主键（由 MongoDB 提供的默认 key\_id） |

| **数据库服务器** | **客户端** |
| --- | --- |
| MySQL/Oracle | MongoDB |
| mysql/sqlplus | mongo |

## 范例文档

下面这个范例展示了一个简单的博客站点的文档结构，它是由逗号分隔的键值对构成的。

{

\_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,

comments: [

{

user:'user1',

message: 'My first comment',

dateCreated: new Date(2011,1,20,2,15),

like: 0

},

{

user:'user2',

message: 'My second comments',

dateCreated: new Date(2011,1,25,7,45),

like: 5

}

]

}

**\_id** 是一个 12 字节长的十六进制数，它保证了每一个文档的唯一性。在插入文档时，需要提供 \_id。如果你不提供，那么 MongoDB 就会为每一文档提供一个唯一的 id。\_id 的头 4 个字节代表的是当前的时间戳，接着的后 3 个字节表示的是机器 id 号，接着的 2 个字节表示 MongoDB 服务器进程 id，最后的 3 个字节代表递增值。

# 优势

任何关系型数据库都采用一种典型的设计模式，展示表的数目以及表之间的关系。然而 MongoDB 却没有关系这个概念。

## MongoDB 相比 RDBMS 的优势

* 模式较少：MongoDB 是一种文档数据库，一个集合可以包含各种不同的文档。每个文档的字段数、内容以及文档大小都可以各不相同。
* 采用单个对象的模式，清晰简洁。
* 没有复杂的连接功能。
* 深度查询功能。MongoDB 支持对文档执行动态查询，使用的是一种不逊色于 SQL 语言的基于文档的查询语言。
* 具有调优功能。
* 易于扩展。MongoDB 非常易于扩展。
* 不需要从应用对象到数据库对象的转换/映射。
* 使用内部存储存储（窗口化）工作集，能够更快地访问数据。

## 为何选择使用 MongoDB

* 面向文档的存储：以 JSON 格式的文档保存数据。
* 任何属性都可以建立索引。
* 复制以及高可扩展性。
* 自动分片。
* 丰富的查询功能。
* 快速的即时更新。
* 来自 MongoDB 的专业支持。

## MongoDB 适用的领域

* 大数据
* 内容管理及交付
* 移动及社会化基础设施
* 用户数据管理
* 数据中心

# 数据模型

MongoDB 中的数据模式非常灵活。同一集合中的文档不需要具有同一字段或结构，集合文档的公用字段可能包含不同类型的数据。

## 设计 MongDB 模式时应注意的问题

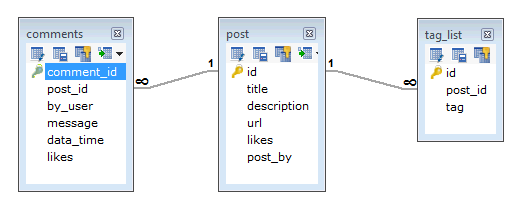
* 根据用户需求来设计模式。
* 如果想一起使用对象，请将这些对象合并到一个文档中，否则要将它们分开（但是要确保不需要连接）。
* 经常复制数据（但要有一定限度），因为与计算时间相比，硬盘空间显得非常便宜。
* 在写入时进行连接，而不能在读取时连接。
* 针对经常发生的用例来设计模式。
* 在模式中实现复杂的聚合。

## 范例

假使一个客户需要为他的博客站点设计一个数据库，让我们来看看 RDBMS 与 MongoDB 在模式设计上的差异。网站需求如下所示：

* 每篇博客都具有唯一的标题、描述以及 URL。
* 每篇博客都具有一个或多个标签。
* 每篇博客都具有发表者的名称，以及喜欢
* 每篇博客都有用户的评论，用户名、消息、日期时间以及评论的喜欢度。
* 每篇博客都可以有 0 个或多个评论。

在 RDBMS 中，设计一个能够满足上述需求的数据库模式至少需要 3 个表。如下图所示。



在 MongoDB 中，设计出来的模式却只有一个集合 post，其结构如下：

{

\_id: POST\_ID

title: TITLE\_OF\_POST,

description: POST\_DESCRIPTION,

by: POST\_BY,

url: URL\_OF\_POST,

tags: [TAG1, TAG2, TAG3],

likes: TOTAL\_LIKES,

comments: [

{

user:'COMMENT\_BY',

message: TEXT,

dateCreated: DATE\_TIME,

like: LIKES

},

{

user:'COMMENT\_BY',

message: TEXT,

dateCreated: DATE\_TIME,

like: LIKES

}

]

}

虽然只是展示数据，在 RDBMS 中需要连接三张表，而在 MongoDB 中则只需要一个集合。

# 创建数据库

## use 命令

MongoDB 用 use + **数据库名称** 的方式来创建数据库。use 会创建一个新的数据库，如果该数据库存在，则返回这个数据库。

### 语法格式

use 语句的基本格式如下：

use DATABASE\_NAME

### 范例

创建一个名为 的数据库，使用 use 语句如下：

>use mydb

switched to db mydb

使用命令 db 检查当前选定的数据库。

>db

mydb

使用命令 show dbs 来检查数据库列表。

>show dbs

local 0.78125GB

test 0.23012GB

刚创建的数据库（mydb）没有出现在列表中。为了让数据库显示出来，至少应该插入一个文档。

>db.movie.insert({"name":"tutorials point"})

>show dbs

local 0.78125GB

mydb 0.23012GB

test 0.23012GB

db:表示的是当前数据表

movie:表示当前数据库的的一个集合,有sql基础的话可以理解为当前数据下的一个表

name:表示该集合下的一个字段

tutorials point:表示name字段的值

在 MongoDB 中，默认的数据库是 test，如果你没有创建任何数据库，那么集合就会保存在 test 数据库中。

# 删除数据库

## dropDatabase() 方法

MongoDB 的 dropDatabase() 命令用于删除已有数据库。

### 语法格式

dropDatabase() 命令的语法格式如下：

db.dropDatabase()

它将删除选定的数据库。如果没有选定要删除的数据库，则它会将默认的 test 数据库删除。

### 范例

首先使用 show dbs 来列出已有的数据库。

>show dbs

local 0.78125GB

mydb 0.23012GB

test 0.23012GB

>

如果想删除新数据库 <mydb>，如下面这样使用 dropDatabase() 方法：

>use mydb

switched to db mydb

>db.dropDatabase()

>{ "dropped" : "mydb", "ok" : 1 }

>

再来看一下数据库列表，确实删除了 <mydb>。

>show dbs

local 0.78125GB

test 0.23012GB

>

# 创建集合

## createCollection() 方法

在 MongoDB 中，创建集合采用 **db.createCollection(name, options)** 方法。

### 语法格式

createCollection() 方法的基本格式如下：

db.createCollection(name, options)

在该命令中，name 是所要创建的集合名称。options 是一个用来指定集合配置的文档。

| **参数** | **类型** | **描述** |
| --- | --- | --- |
| name | 字符串 | 所要创建的集合名称 |
| options | 文档 | 可选。指定有关内存大小及索引的选项 |

参数 options 是可选的，所以你必须指定的只有集合名称。下表列出了所有可用选项：

| **字段** | **类型** | **描述** |
| --- | --- | --- |
| capped | 布尔 | （可选）如果为 true，则创建固定集合。固定集合是指有着固定大小的集合，当达到最大值时，它会自动覆盖最早的文档。 **当该值为 true 时，必须指定 size 参数。** |
| autoIndexId | 布尔 | （可选）如为 true，自动在 \_id 字段创建索引。默认为 false。 |
| size | 数值 | （可选）为固定集合指定一个最大值（以字节计）。 **如果 capped 为 true，也需要指定该字段。** |
| max | 数值 | （可选）指定固定集合中包含文档的最大数量。 |

在插入文档时，MongoDB 首先检查固定集合的 size 字段，然后检查 max 字段。

### 范例

不带参数的 createCollection() 方法的基本格式为：

>use test

switched to db test

>db.createCollection("mycollection")

{ "ok" : 1 }

>

可以使用 show collections 来查看创建了的集合。

>show collections

mycollection

system.indexes

下面是带有几个关键参数的 createCollection() 的用法：

>db.createCollection("mycol", { capped : true, autoIndexID : true, size : 6142800, max : 10000 } )

{ "ok" : 1 }

>

在 MongoDB 中，你不需要创建集合。当你插入一些文档时，MongoDB 会自动创建集合。

>db.tutorialspoint.insert({"name" : "tutorialspoint"})

>show collections

mycol

mycollection

system.indexes

tutorialspoint

>

# 删除集合

## drop() 方法

MongoDB 利用 db.collection.drop() 来删除数据库中的集合。

### 语法格式

drop() 命令的基本格式如下：

db.COLLECTION\_NAME.drop()

### 范例

首先检查在数据库 mydb 中已有集合：

>use mydb

switched to db mydb

>show collections

mycol

mycollection

system.indexes

tutorialspoint

>

接着删除集合 mycollection。

>db.mycollection.drop()

true

>

再次检查数据库中的现有集合：

>show collections

mycol

system.indexes

tutorialspoint

>

如果成功删除选定集合，则 drop() 方法返回 true，否则返回 false。

# 数据类型

MongoDB 支持如下数据类型：

* **String**：字符串。存储数据常用的数据类型。在 MongoDB 中，UTF-8 编码的字符串才是合法的。
* **Integer**：整型数值。用于存储数值。根据你所采用的服务器，可分为 32 位或 64 位。
* **Boolean**：布尔值。用于存储布尔值（真/假）。
* **Double**：双精度浮点值。用于存储浮点值。
* **Min/Max keys**：将一个值与 BSON（二进制的 JSON）元素的最低值和最高值相对比。
* **Arrays**：用于将数组或列表或多个值存储为一个键。
* **Timestamp**：时间戳。记录文档修改或添加的具体时间。
* **Object**：用于内嵌文档。
* **Null**：用于创建空值。
* **Symbol**：符号。该数据类型基本上等同于字符串类型，但不同的是，它一般用于采用特殊符号类型的语言。
* **Date**：日期时间。用 UNIX 时间格式来存储当前日期或时间。你可以指定自己的日期时间：创建 Date 对象，传入年月日信息。
* **Object ID**：对象 ID。用于创建文档的 ID。
* **Binary Data**：二进制数据。用于存储二进制数据。
* **Code**：代码类型。用于在文档中存储 JavaScript 代码。
* **Regular expression**：正则表达式类型。用于存储正则表达式。

# 插入文档

## insert() 方法

要想将数据插入 MongoDB 集合中，需要使用 insert() 或 save() 方法。

### 语法格式

insert() 方法的基本格式为：

>db.COLLECTION\_NAME.insert(document)

### 范例 1

>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

})

mycol 是上一节所创建的集合的名称。如果数据库中不存在该集合，那么 MongoDB 会创建该集合，并向其中插入文档。

在插入的文档中，如果我们没有指定 \_id 参数，那么 MongoDB 会自动为文档指定一个唯一的 ID。

\_id 是一个 12 字节长的 16 进制数，这 12 个字节的分配如下：

\_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incrementer)

为了，你可以将用 insert() 方法传入一个文档数组，范例如下：

### 范例 2

>db.post.insert([

{

title: 'MongoDB Overview',

description: 'MongoDB is no sql database',

by: 'tutorials point',

url: 'http://www.tutorialspoint.com',

tags: ['mongodb', 'database', 'NoSQL'],

likes: 100

},

{

title: 'NoSQL Database',

description: 'NoSQL database doesn't have tables',

by: 'tutorials point',

url: 'http://www.tutorialspoint.com',

tags: ['mongodb', 'database', 'NoSQL'],

likes: 20,

comments: [

{

user:'user1',

message: 'My first comment',

dateCreated: new Date(2013,11,10,2,35),

like: 0

}

]

}

])

也可以用 db.post.save(document) 插入文档。如果没有指定文档的 \_id，那么 save() 就和 insert() 完全一样了。

一、使用save()函数，如果原来的对象不存在，那他们都可以向collection里插入数据，如果已经存在，save会调用update更新里面的记录，而insert则会忽略操作

二、insert可以一次性插入一个列表，而不用遍历，效率高， save则需要遍历列表，一个个插入。

save方法实际上执行的是upsert,只是按id查找,所有字段更新,如果save的文档中没有指定id则save方法实际执行的是insert方法并自动生成id

# 查询文档

## find() 方法

要想查询 MongoDB 集合中的数据，使用 find() 方法。

db.collection.find({}).count()

- 查询所有结果的数量

### 语法格式

find() 方法的基本格式为：

>db.COLLECTION\_NAME.find()

find() 方法会以非结构化的方式来显示所有文档。

## pretty() 方法

用格式化方式显示结果，使用的是 pretty() 方法。

### 语法格式

>db.mycol.find().pretty()

### 范例

>db.mycol.find().pretty()

{

"\_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"

}

>

除了 find() 方法之外，还有一个 findOne() 方法，它只返回一个文档。

## MongoDB 中类似于 WHERE 子句的语句

如果想要基于一些条件来查询文档，可以使用下列操作。

| **操作** | **格式** | **范例** | **RDBMS中的类似语句** |
| --- | --- | --- | --- |
| 等于 | {<key>:<value>} | db.mycol.find({"by":"tutorials point"}).pretty() | where by = 'tutorials point' |
| 小于 | {<key>:{$lt:<value>}} | db.mycol.find({"likes":{$lt:50}}).pretty() | where likes < 50 |
| 小于或等于 | {<key>:{$lte:<value>}} | db.mycol.find({"likes":{$lte:50}}).pretty() | where likes <= 50 |
| 大于 | {<key>:{$gt:<value>}} | db.mycol.find({"likes":{$gt:50}}).pretty() | where likes > 50 |
| 大于或等于 | {<key>:{$gte:<value>}} | db.mycol.find({"likes":{$gte:50}}).pretty() | where likes >= 50 |
| 不等于 | {<key>:{$ne:<value>}} | db.mycol.find({"likes":{$ne:50}}).pretty() | where likes != 50 |

## MongoDB 中的 And 条件

### 语法格式

在 find() 方法中，如果传入多个键，并用逗号(,)分隔它们，那么 MongoDB 会把它看成是 **AND** 条件。AND 条件的基本语法格式为：

>db.mycol.find({key1:value1, key2:value2}).pretty()

### 范例

下例将展示所有由 “tutorials point” 发表的标题为 “MongoDB Overview” 的教程。

>db.mycol.find({"by":"tutorials point","title": "MongoDB Overview"}).pretty()

{

"\_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"

}

>

对于上例这种情况，RDBMS 采用的 WHERE 子句将会是：**where by='tutorials point' AND title='MongoDB Overview'**。你可以在 find 子句中传入任意的键值对。

## MongoDB 中的 OR 条件

### 语法格式

若基于 **OR** 条件来查询文档，可以使用关键字 **$or**。 OR 条件的基本语法格式为：

>db.mycol.find(

{

$or: [

{key1: value1}, {key2:value2}

]

}

).pretty()

### 范例

下例将展示所有由 “tutorials point” 发表的或标题为 “MongoDB Overview” 的教程。

>db.mycol.find({$or:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty()

{

"\_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"

}

>

## 结合使用 AND 与 OR 条件

### 范例

下例所展示文档的条件为：喜欢数大于 100，标题是 “MongoDB Overview”，或者是由 “tutorials point” 所发表的。响应的 SQL WHERE 子句为：**where likes>10 AND (by = 'tutorials point' OR title = 'MongoDB Overview')**

>db.mycol.find({"likes": {$gt:10}, $or: [{"by": "tutorials point"},{"title": "MongoDB Overview"}]}).pretty()

{

"\_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"

}

>

# 更新文档

MongoDB 中的 update() 与 save() 方法都能用于更新集合中的文档。update() 方法更新已有文档中的值，而 save() 方法则是用传入该方法的文档来替换已有文档。

## update() 方法

update() 方法更新已有文档中的值。

### 语法格式

update() 方法基本格式如下：

>db.COLLECTION\_NAME.update(SELECTIOIN\_CRITERIA, UPDATED\_DATA)

- 如果需要修改指定的属性，而不是替换需要使用“修改操作符”来完成修改

$set 可以用来修改文档中的指定属性

$unset 可以用来删除文档的指定属性

db.student.update({\_id:1},{**$unset**:{"new":""}})

### 范例

假如 mycol 集合中有下列数据：

{ "\_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

下面的例子将把文档原标题 'MongoDB Overview' 替换为新的标题 'New MongoDB Tutorial'。

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

>db.mycol.find()

{ "\_id" : ObjectId(5983548781331adf45ec5), "title":"New MongoDB Tutorial"}

{ "\_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

>

MongoDB 默认只更新单个文档，要想更新多个文档，需要把参数 multi 设为 true。

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

## save() 方法

save() 方法利用传入该方法的文档来替换已有文档。

### 语法格式

save() 方法基本语法格式如下：

>db.COLLECTION\_NAME.save({\_id:ObjectId(),NEW\_DATA})

### 范例

下例用 \_id 为 '5983548781331adf45ec7' 的文档代替原有文档。

>db.mycol.save(

{

"\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point New Topic", "by":"Tutorials Point"

}

)

>db.mycol.find()

{ "\_id" : ObjectId(5983548781331adf45ec5), "title":"Tutorials Point New Topic", "by":"Tutorials Point"}

{ "\_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

>

# 删除文档

## remove() 方法

MongoDB 利用 remove() 方法 清除集合中的文档。它有 2 个可选参数：

* deletion criteria：（可选）删除文档的标准。
* justOne：（可选）如果设为 true 或 1，则只删除一个文档。

### 语法格式

remove() 方法的基本语法格式如下所示：

>db.COLLECTION\_NAME.remove(DELLETION\_CRITTERIA)

### 范例

假如 mycol 集合中包含下列数据：

{ "\_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

下面我们将删除其中所有标题为 'MongoDB Overview' 的文档。

>db.mycol.remove({'title':'MongoDB Overview'})

>db.mycol.find()

{ "\_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

>

## 只删除一个文档

如果有多个记录，而你只想删除第一条记录，那么就设置 remove() 方法中的 justOne 参数：

>db.COLLECTION\_NAME.remove(DELETION\_CRITERIA,1)

## 删除所有文档

如果没有指定删除标准，则 MongoDB 会将集合中所有文档都予以删除。**这等同于 SQL 中的 truncate 命令**。

>db.mycol.remove({})

>db.mycol.find()

>

# 映射

在 MongoDB 中，映射（Projection）指的是只选择文档中的必要数据，而非全部数据。如果文档有 5 个字段，而你只需要显示 3 个，则只需选择 3 个字段即可。

## find() 方法

MongoDB 的查询文档曾介绍过 find() 方法，它可以利用 AND 或 OR 条件来获取想要的字段列表。在 MongoDB 中执行 find() 方法时，显示的是一个文档的所有字段。要想限制，可以利用 0 或 1 来设置字段列表。1 用于显示字段，0 用于隐藏字段。

### 语法格式

带有映射的 find() 方法的基本语法格式为：

>db.COLLECTION\_NAME.find({},{KEY:1})

### 范例

假如 mycol 集合拥有下列数据：

{ "\_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

下例将在查询文档时显示文档标题。

>db.mycol.find({},{"title":1,\_id:0})

{"title":"MongoDB Overview"}

{"title":"NoSQL Overview"}

{"title":"Tutorials Point Overview"}

>

注意：在执行 find() 方法时，\_id 字段是一直显示的。如果不想显示该字段，则可以将其设为 0。

# 限制记录

## limit() 方法

要想限制 MongoDB 中的记录，可以使用 limit() 方法。limit() 方法接受一个数值类型的参数，其值为想要显示的文档数。

### 语法格式

limit() 方法的基本语法格式为：

>db.COLLECTION\_NAME.find().limit(NUMBER)

### 范例

假设 mycol 集合拥有下列数据：

{ "\_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

下例将在查询文档时只显示 2 个文档。

>db.mycol.find({},{"title":1,\_id:0}).limit(2)

{"title":"MongoDB Overview"}

{"title":"NoSQL Overview"}

>

如果未指定 limit() 方法中的数值参数，则将显示该集合内的所有文档。

## skip() 方法

### 语法格式

skip() 方法基本语法格式为：

>db.COLLECTION\_NAME.find().skip(NUMBER).limit(NUMBER)

### 范例

下例将只显示第二个文档：

>db.mycol.find({},{"title":1,\_id:0}).skip(1) .limit(1)

{"title":"NoSQL Overview"}

>

注意：skip() 方法中的默认值为 0。

# 记录排序

## sort() 方法

MongoDB 中的文档排序是通过 sort() 方法来实现的。sort() 方法可以通过一些参数来指定要进行排序的字段，并使用 1 和 -1 来指定排序方式，其中 1 表示升序，而 -1 表示降序。

### 格式

sort() 方法基本格式为：

>db.COLLECTION\_NAME.find().sort({KEY:1})

### 范例

假设集合 mycol 包含下列数据：

{ "\_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

下面的范例将显示按照降序排列标题的文档。

>db.mycol.find({},{"title":1,\_id:0}).sort({"title":-1})

{"title":"Tutorials Point Overview"}

{"title":"NoSQL Overview"}

{"title":"MongoDB Overview"}

>

注意，如果不指定排序规则，sort() 方法将按照升序排列显示文档。

# 索引

索引能够实现高效地查询。没有索引，MongoDB 就必须扫描集合中的所有文档，才能找到匹配查询语句的文档。这种扫描毫无效率可言，需要处理大量的数据。

索引是一种特殊的数据结构，将一小块数据集保存为容易遍历的形式。索引能够存储某种特殊字段或字段集的值，并按照索引指定的方式将字段值进行排序。

## ensureIndex() 方法

要想创建索引，需要使用 MongoDB 的 ensureIndex() 方法。

mongodb使用createIndex()和ensureIndex()方法来创建索引，前者用于3.0及以上版本，后者用于3.0以下版本。

### 语法格式

ensureIndex() 方法的基本语法格式为：

>db.COLLECTION\_NAME.ensureIndex({KEY:1})

这里的 key 是想创建索引的字段名称，1 代表按升序排列字段值。-1 代表按降序排列。

### 范例

>db.mycol.ensureIndex({"title":1})

>

可以为 ensureIndex() 方法传入多个字段，从而为多个字段创建索引。

>db.mycol.ensureIndex({"title":1,"description":-1})

>

ensureIndex() 方法也可以接受一些可选参数，如下所示：

| **参数** | **类型** | **描述** |
| --- | --- | --- |
| background | 布尔值 | 在后台构建索引，从而不干扰数据库的其他活动。取值为 true 时，代表在后台构建索引。默认值为 false |
| unique | 布尔值 | 创建一个唯一的索引，从而当索引键匹配了索引中一个已存在值时，集合不接受文档的插入。取值为 true 代表创建唯一性索引。默认值为 false。 |
| name | 字符串 | 索引名称。如果未指定，MongoDB 会结合索引字段名称和排序序号，生成一个索引名称。 |
| dropDups | 布尔值 | 在可能有重复的字段内创建唯一性索引。MongoDB 只在某个键第一次出现时进行索引，去除该键后续出现时的所有文档。 |
| sparse | 布尔值 | 如果为 true，索引只引用带有指定字段的文档。这些索引占据的空间较小，但在一些情况下的表现也不同（特别是排序）。默认值为 false。 |
| expireAfterSeconds | 整型值 | 指定一个秒数值，作为 TTL 来控制 MongoDB 保持集合中文档的时间。 |
| v | 索引版本 | 索引版本号。默认的索引版本跟创建索引时运行的 MongoDB 版本号有关。 |
| weights | 文档 | 数值，范围从 1 到 99, 999。表示就字段相对于其他索引字段的重要性。 |
| default\_language | 字符串 | 对文本索引而言，用于确定停止词列表，以及词干分析器（stemmer）与断词器（tokenizer）的规则。默认值为 english。 |
| language\_override | 字符串 | 对文本索引而言，指定了文档所包含的字段名，该语言将覆盖默认语言。默认值为 language。 |

## 创建索引

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

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

例2：给name字段创建倒序索引

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

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 2,

"numIndexesAfter" : 3,

"ok" : 1

}

例3：给name，age字段创建组合索引

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

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 3,

"numIndexesAfter" : 4,

"ok" : 1

}

例4：在后台给age字段创建索引

> db.users.createIndex({age:1},{background:1})

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 4,

"numIndexesAfter" : 5,

"ok" : 1

}

在后台创建索引的原因：  
在前台创建索引期间会锁定数据库，会导致其它操作无法进行数据读写，在后台创建索引是，会定期释放写锁，从而保证其它操作的运行，但是后台操作会在耗时更长，尤其是在频繁进行写入的服务器上。

## 查看索引

MongoDB提供的查看索引信息的方法：  
getIndexes()方法可以用来查看集合的所有索引，  
getIndexKeys()方法查看索引键。  
totalIndexSize()查看集合索引的总大小，  
getIndexSpecs()方法查看集合各索引的详细信息

例1： getIndexes()的用法

> db.users.getIndexes()

[

{

"v" : 1,

"key" : {

"\_id" : 1

},

"name" : "\_id\_",

"ns" : "test1.users"

},

{

"v" : 1,

"key" : {

"name" : 1

},

"name" : "name\_1",

"ns" : "test1.users"

},

{

"v" : 1,

"key" : {

"name" : -1

},

"name" : "name\_-1",

"ns" : "test1.users"

},

{

"v" : 1,

"key" : {

"name" : 1,

"age" : 1

},

"name" : "name\_1\_age\_1",

"ns" : "test1.users"

},

{

"v" : 1,

"key" : {

"age" : 1

},

"name" : "age\_1",

"ns" : "test1.users",

"background" : 1

}

]

例2：getIndexKeys()的用法

> db.users.getIndexKeys()

[

{

"\_id" : 1

},

{

"name" : 1

},

{

"name" : -1

},

{

"name" : 1,

"age" : 1

},

{

"age" : 1

}

]

例3：totalIndexSize()的用法

> **db**.users.totalIndexSize()

81920

例4：getIndexSpecs()的用法

> db.users.getIndexSpecs()

[

{

"v" : 1,

"key" : {

"\_id" : 1

},

"name" : "\_id\_",

"ns" : "test1.users"

},

{

"v" : 1,

"key" : {

"name" : 1

},

"name" : "name\_1",

"ns" : "test1.users"

},

{

"v" : 1,

"key" : {

"name" : -1

},

"name" : "name\_-1",

"ns" : "test1.users"

},

{

"v" : 1,

"key" : {

"name" : 1,

"age" : 1

},

"name" : "name\_1\_age\_1",

"ns" : "test1.users"

},

{

"v" : 1,

"key" : {

"age" : 1

},

"name" : "age\_1",

"ns" : "test1.users",

"background" : 1

}

]

## 删除索引

不再需要的索引，我们可以将其删除，mongodb提供两种删除索引的方法：  
dropIndex( name )方法用于删除指定的索引  
dropIndexes()方法用于删除全部的索引

例1:dropIndex()的用法

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

{ "nIndexesWas" : 5, "ok" : 1 }

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

{ "nIndexesWas" : 4, "ok" : 1 }

> db.users.getIndexSpecs()

[

{

"v" : 1,

"key" : {

"\_id" : 1

},

"name" : "\_id\_",

"ns" : "test1.users"

},

{

"v" : 1,

"key" : {

"name" : -1

},

"name" : "name\_-1",

"ns" : "test1.users"

},

{

"v" : 1,

"key" : {

"age" : 1

},

"name" : "age\_1",

"ns" : "test1.users",

"background" : 1

}

]

我们可以看到,name字段的索引和name与age字段的组合索引皆被删除

例2:dropIndexes()的用法

> db.users.**dropIndexes**()

{

"nIndexesWas" : 3,

"msg" : "non-\_id indexes dropped for collection",

"ok" : 1

}

> db.users.getIndexSpecs()

[

{

"v" : 1,

"key" : {

"\_id" : 1

},

"name" : "\_id\_",

"ns" : "test1.users"

}

]

在使用了dropIndexes()方法后,我们之前建的所有索引都被删除掉了

# Mongoose

## 简介

*/\*  
 1.下载安装Mongoose  
 npm i mongoose --save  
 2.在项目中引入mongoose  
 var mongoose = require("mongoose");  
 3.连接MongoDB数据库  
 mongoose.connect('mongodb://数据库的ip地址:端口号/数据库名', { useMongoClient: true});  
 - 如果端口号是默认端口号（27017） 则可以省略不写  
  
 4.断开数据库连接(一般不需要调用)  
 - MongoDB数据库，一般情况下，只需要连接一次，连接一次以后，除非项目停止服务器关闭，否则连接一般不会断开  
 mongoose.disconnect()  
  
 - 监听MongoDB数据库的连接状态  
 - 在mongoose对象中，有一个属性叫做connection，该对象表示的就是数据库连接  
 通过监视该对象的状态，可以来监听数据库的连接与断开  
  
 数据库连接成功的事件  
 mongoose.connection.once("open",function(){});  
  
 数据库断开的事件  
 mongoose.connection.once("close",function(){});  
  
 Schema  
 Model  
 Document  
  
 \*/  
  
//引入***var *mongoose*** = *require*(**"mongoose"**);  
*//连接数据库****mongoose***.connect(**"mongodb://127.0.0.1/mongoose\_test"** , { **useMongoClient**: **true**});  
  
***mongoose***.connection.once(**"open"**,**function**(){  
 ***console***.log(**"数据库连接成功~~~"**);  
});  
  
***mongoose***.connection.once(**"close"**,**function**(){  
 ***console***.log(**"数据库连接已经断开~~~"**);  
});  
  
*//断开数据库连接****mongoose***.disconnect();

## Schema和model

**var *mongoose*** = *require*(**"mongoose"**);  
***mongoose***.connect(**"mongodb://127.0.0.1/mongoose\_test"**,{**useMongoClient**:**true**});  
***mongoose***.connection.once(**"open"**,**function** () {  
 ***console***.log(**"数据库连接成功~~~"**);  
});  
  
  
*//将mongoose.Schema 赋值给一个变量***var *Schema*** = ***mongoose***.Schema;  
  
*//创建Schema（模式）对象***var *stuSchema*** = **new *Schema***({  
  
 **name**:String,  
 **age**:Number,  
 **gender**:{  
 **type**:String,  
 **default**:**"female"** },  
 **address**:String  
  
});  
  
*//通过Schema来创建Model  
//Model代表的是数据库中的集合，通过Model才能对数据库进行操作  
//mongoose.model(modelName, schema):  
//modelName 就是要映射的集合名 mongoose会自动将集合名变成复数***var *StuModel*** = ***mongoose***.**model**(**"student"** , ***stuSchema***);  
  
*//向数据库中插入一个文档  
//StuModel.create(doc, function(err){});****StuModel***.create({  
 **name**:**"白骨精"**,  
 **age**:16,  
 **address**:**"白骨洞"**},**function** (err) {  
 **if**(!err){  
 ***console***.log(**"插入成功~~~"**);  
 }  
});

## Model的方法

**var *mongoose*** = *require*(**"mongoose"**);  
***mongoose***.connect(**"mongodb://127.0.0.1/mongoose\_test"**,{**useMongoClient**:**true**});  
***mongoose***.connection.once(**"open"**,**function** () {  
 ***console***.log(**"数据库连接成功~~~"**);  
});  
  
**var *Schema*** = ***mongoose***.Schema;  
  
**var *stuSchema*** = **new *Schema***({  
  
 **name**:String,  
 **age**:Number,  
 **gender**:{  
 **type**:String,  
 **default**:**"female"** },  
 **address**:String  
  
});  
  
**var *StuModel*** = ***mongoose***.**model**(**"student"** , ***stuSchema***);  
*/\*  
 - 有了Model，我们就可以来对数据库进行增删改查的操作了  
  
 Model.create(doc(s), [callback])  
 - 用来创建一个或多个文档并添加到数据库中  
 - 参数：  
 doc(s) 可以是一个文档对象，也可以是一个文档对象的数组  
 callback 当操作完成以后调用的回调函数  
  
 查询的：  
 Model.find(conditions, [projection], [options], [callback])  
 - 查询所有符合条件的文档 总会返回一个数组  
 Model.findById(id, [projection], [options], [callback])  
 - 根据文档的id属性查询文档  
 Model.findOne([conditions], [projection], [options], [callback])  
 - 查询符合条件的第一个文档 总和返回一个具体的文档对象  
  
 conditions 查询的条件  
 projection 投影 需要获取到的字段  
 - 两种方式  
 {name:1,\_id:0}  
 "name -\_id"  
 options 查询选项（skip limit）  
 {skip:3 , limit:1}  
 callback 回调函数，查询结果会通过回调函数返回  
 回调函数必须传，如果不传回调函数，压根不会查询  
  
 \*/  
  
/\*StuModel.find({name:"唐僧"},function (err , docs) {  
 if(!err){  
 console.log(docs);  
 }  
});\*/  
  
/\*StuModel.find({},{name:1 , \_id:0},function (err , docs) {  
 if(!err){  
 console.log(docs);  
 }  
});\*/  
  
/\*StuModel.find({},"name age -\_id", {skip:3 , limit:1} , function (err , docs) {  
 if(!err){  
 console.log(docs);  
 }  
});\*/  
  
/\*StuModel.findOne({} , function (err , doc) {  
 if(!err){  
 console.log(doc);  
 }  
});\*/  
  
/\*StuModel.findById("59c4c3cf4e5483191467d392" , function (err , doc) {  
 if(!err){  
 //console.log(doc);  
 //通过find()查询的结果，返回的对象，就是Document，文档对象  
 //Document对象是Model的实例  
 console.log(doc instanceof StuModel);  
 }  
});\*/  
  
  
  
/\*StuModel.create([  
 {  
 name:"沙和尚",  
 age:38,  
 gender:"male",  
 address:"流沙河"  
 }  
  
],function (err) {  
 if(!err){  
 console.log(arguments);  
 }  
});\*/  
  
  
/\*  
 修改  
 Model.update(conditions, doc, [options], [callback])  
 Model.updateMany(conditions, doc, [options], [callback])  
 Model.updateOne(conditions, doc, [options], [callback])  
 - 用来修改一个或多个文档  
 - 参数：  
 conditions 查询条件  
 doc 修改后的对象  
 options 配置参数  
 callback 回调函数  
 Model.replaceOne(conditions, doc, [options], [callback])  
\* \*/  
  
//修改唐僧的年龄为20  
/\*StuModel.updateOne({name:"唐僧"},{$set:{age:20}},function (err) {  
 if(!err){  
 console.log("修改成功");  
 }  
});\*/  
  
/\*  
 删除：  
 Model.remove(conditions, [callback])  
 Model.deleteOne(conditions, [callback])  
 Model.deleteMany(conditions, [callback])  
 \*/  
/\*StuModel.remove({name:"白骨精"},function (err) {  
 if(!err){  
 console.log("删除成功~~");  
 }  
});\*/  
  
/\*  
 Model.count(conditions, [callback])  
 - 统计文档的数量的  
 \*/****StuModel***.count({},**function** (err , count) {  
 **if**(!err){  
 ***console***.log(count);  
 }  
});

## Document的方法

**var *mongoose*** = *require*(**"mongoose"**);  
***mongoose***.connect(**"mongodb://127.0.0.1/mongoose\_test"**,{**useMongoClient**:**true**});  
***mongoose***.connection.once(**"open"**,**function** () {  
 ***console***.log(**"数据库连接成功~~~"**);  
});  
  
**var *Schema*** = ***mongoose***.Schema;  
  
**var *stuSchema*** = **new *Schema***({  
  
 **name**:String,  
 **age**:Number,  
 **gender**:{  
 **type**:String,  
 **default**:**"female"** },  
 **address**:String  
  
});  
  
**var *StuModel*** = ***mongoose***.**model**(**"student"** , ***stuSchema***);  
*/\*  
 Document 和 集合中的文档一一对应 ， Document是Model的实例  
 通过Model查询到结果都是Document  
 \*/  
  
//创建一个Document***var *stu*** = **new *StuModel***({  
 **name**:**"奔波霸"**,  
 **age**:48,  
 **gender**:**"male"**,  
 **address**:**"碧波潭"**});  
  
*/\*  
 document的方法  
 Model#save([options], [fn])  
 \*/  
/\*  
stu.save(function (err) {  
 if(!err){  
 console.log("保存成功~~~");  
 }  
});\*/****StuModel***.findOne({},**function** (err , doc) {  
 **if**(!err){  
 */\*  
 update(update,[options],[callback])  
 - 修改对象  
 remove([callback])  
 - 删除对象  
  
 \*/  
 //console.log(doc);  
 /\*doc.update({$set:{age:28}},function (err) {  
 if(!err){  
 console.log("修改成功~~~");  
 }  
 });\*/  
  
 /\*doc.age = 18;  
 doc.save();\*/  
  
 /\*doc.remove(function (err) {  
 if(!err){  
 console.log("大师兄再见~~~");  
 }  
 });\*/  
  
  
 /\*  
 get(name)  
 - 获取文档中的指定属性值  
 set(name , value)  
 - 设置文档的指定的属性值  
 id  
 - 获取文档的\_id属性值  
 toJSON() \*\*\*\*\*\*  
 - 转换为一个JSON对象  
  
 toObject()  
 - 将Document对象转换为一个普通的JS对象  
 转换为普通的js对象以后，注意所有的Document对象的方法或属性都不能使用了  
  
 \*/  
 //console.log(doc.get("age"));  
 //console.log(doc.age);  
  
 //doc.set("name","猪小小");  
 //doc.name = "hahaha";  
  
 //console.log(doc.\_id);  
 //var j = doc.toJSON();  
 //console.log(j);  
  
 //var o = doc.toObject();  
  
 //console.log(o);* doc = doc.toObject();  
  
 **delete** doc.address;  
  
 ***console***.log(doc.\_id);  
  
 }  
});