**Mongodb数据库学习**

# Nosql数据库介绍

mongodb是当下最流行的NoSQL数据库中的一种。

## NoSQL的诞生

随着互联网的发展，当我们把一台服务器一台服务器变成两台服务器，当我们开始建立数据备份，当我们需要加一个缓冲层，来调整所有的查询，投入更多的硬件。最后，需要将数据切分多个集群上，并重构大量的应用逻辑以适应这种切分。不久之后，你就会发现被自己数月前的设计数据结构限制住了。

随着web2.0的兴起，关系型数据库本身无法克服的缺陷越来越明显，主要表现为如下几点。

* + 对数据高并发读写的需求
  + 对海量数据的高效率存储和访问的需求。
  + 对数据库的高可扩展性和高可用性的需求。

NoSQL是Not only SQL的缩写，NoSQL不使用SQL作为查询语言。其数据存储可以不需要固定的表格模式，也经常避免使用SQL的join操作，一般有水平可扩展性的特征。

## NoSQL的历史

1. NoSQL一词最早出现在1998年，是Carlo Strozzi开发的一个轻量、开源、不提供SQL功能的关系数据库。
2. 2009年，Last.fm的Johan Oskarsson发起了一次关于分布式开源数据库的讨论，来自Rackspace的Eric Evans再次提出了NoSQL概念，这时的NoSQL主要是指非关系型、分布式、不提供数据库设计模式。
3. 2009年趋势高涨，被定为“非关系型的”数据存储，相对于关系型数据库运用，这一概念无疑是一种全新思维的注入。

## NoSQL和SQL的区别

NoSQL具有如下几点

优点：

* 高并发读写
* 海量数据存储
* 高可扩展性
* 高可用性

缺点：

* + - 缺乏事务一致性
    - 缺乏读写实时性
    - 不支持复杂查询

## NoSQL应用场景

NoSQL数据库在以下的这几种情况下比较适用：

1. 数据模型比较简单；
2. 需要灵活性更强的IT系统；
3. 对数据库性能要求较高；
4. 不需要高度的数据一致性；
5. 对于给定key，比较容易映射复杂值的环境。

## NoSQL分类

### 键值(Key-Value)存储数据库

这一类数据库主要会使用到一个哈希表，这个表中有一个特定的键和一个指针指向特定的数据。Key/value模型对于IT系统来说的优势在于简单、易部署。但是如果DBA只对部分值进行查询或更新的时候，Key/value就显得效率低下了。举例如：Tokyo Cabinet/Tyrant, Redis, Voldemort, Oracle BDB.

### 列存储数据库。

这部分数据库通常是用来应对分布式存储的海量数据。键仍然存在，但是它们的特点是指向了多个列。这些列是由列家族来安排的。如：Cassandra, HBase, Riak.

### 文档型数据库

文档型数据库的灵感是来自于Lotus Notes办公软件的，而且它同第一种键值存储相类似。该类型的数据模型是版本化的文档，半结构化的文档以特定的格式存储，比如JSON。文档型数据库可 以看作是键值数据库的升级版，允许之间嵌套键值。而且文档型数据库比键值数据库的查询效率更高。如：CouchDB, MongoDb. 国内也有文档型数据库SequoiaDB，已经开源。

### 图形(Graph)数据库

图形结构的数据库同其他行列以及刚性结构的SQL数据库不同，它是使用灵活的图形模型，并且能够扩展到多个服务器上。

NoSQL数据库没有标准的查询语言(SQL)，因此进行数据库查询需要制定数据模型。许多NoSQL数据库都有REST式的数据接口或者查询API。如：Neo4J, InfoGrid, Infinite Graph.

# Mongodb介绍

MongoDB由位于纽约的一个名为10gen的组织开发，现在被称为MongoDB Inc.，它最初被开发为PAAS(平台即服务)。 2009年晚些时候，它被作为一个由MongoDB公司维护和支持的开源数据库服务器在市场上引入。

MongoDB的第一个真正产品是从2010年3月发布的MongoDB 1.4版本开始的。2014年1月10日发布的稳定版本：MongoDB2.4.9。

Mongodb由c++语言编写而成，跨平台的话，需要在对应系统进行编译，编码只使用UTF-8

BSON是由10gen开发的一个数据格式，目前主要用于MongoDB中，是MongoDB的数据存储格式。BSON基于JSON格式，选择JSON进行改造的原因主要是JSON的通用性及JSON的schemaless的特性。

BSON主要会实现以下三点目标：

1.更快的遍历速度

对JSON格式来说，太大的JSON结构会导致数据遍历非常慢。在JSON中，要对一个文档进行数据读取，需要对此文档进行扫描才行，需要进行麻烦的数据结构匹配，比如括号的匹配，而BSON对JSON的一大改进就是，它会将JSON的每一个元素的长度存在元素的头部，这样你只需要读取到元素长度就能直接seek到指定的点上进行读取了。

2.操作更简易

对JSON来说，数据存储是无类型的，比如你要修改基本一个值，从9到10，由于从一个字符变成了两个，所以可能其后面的所有内容都需要往后移一位才可以。而使用BSON，你可以指定这个列为数字列，那么无论数字从9长到10还是100，我们都只是在存储数字的那一位上进行修改，不会导致数据总长变大。当然，在MongoDB中，如果数字从整形增大到长整型，还是会导致数据总长变大的。

3.增加了额外的数据类型

JSON是一个很方便的数据交换格式，但是其类型比较有限。BSON在其基础上增加了“byte array”数据类型。这使得二进制的存储不再需要先base64转换后再存成JSON。大大减少了计算开销和数据大小。

当然，在有的时候，BSON相对JSON来说也并没有空间上的优势，比如对{“field”:7}，在JSON的存储上7只使用了一个字节，而如果用BSON，那就是至少4个字节（32位）

特点：

* MongoDB 是一个面向文档存储的数据库，操作起来比较简单和容易。
* 你可以在MongoDB记录中设置任何属性的索引 (如：FirstName="Sameer",Address="8 Gandhi Road")来实现更快的排序。
* 你可以通过本地或者网络创建数据镜像，这使得MongoDB有更强的扩展性。
* 如果负载的增加（需要更多的存储空间和更强的处理能力） ，它可以分布在计算机网络中的其他节点上这就是所谓的分片。
* Mongo支持丰富的查询表达式。查询指令使用JSON形式的标记，可轻易查询文档中内嵌的对象及数组。
* MongoDb 使用update()命令可以实现替换完成的文档（数据）或者一些指定的数据字段 。
* Mongodb中的Map/reduce主要是用来对数据进行批量处理和聚合操作。
* Map和Reduce。Map函数调用emit(key,value)遍历集合中所有的记录，将key与value传给Reduce函数进行处理。
* Map函数和Reduce函数是使用Javascript编写的，并可以通过db.runCommand或mapreduce命令来执行MapReduce操作。
* GridFS是MongoDB中的一个内置功能，可以用于存放大量小文件。
* MongoDB允许在服务端执行脚本，可以用Javascript编写某个函数，直接在服务端执行，也可以把函数的定义存储在服务端，下次直接调用即可。
* MongoDB支持各种编程语言:RUBY，PYTHON，JAVA，C++，PHP，C#等多种语言。
* MongoDB安装简单。

ObjectId介绍：

MongoDB确实是最像关系型数据库的NoSQL，这在它主键设计上可以体现的出来，它并没有采用自动增长主键，因为在分布式服务器之间做数据同步很麻烦，而是采用了一种ObjectId的方式，它生成方便，占用空间比long多了4个字节，（12个字节）在数据表现层面也说的过去，它是一种以时间，机器，进程和自增几个因素组合的方式来体现的，可以近似看成是按时间的先后进行排序的，使用16进制表示

4字节时间戳+3字节机器码+2字节PID+3字节计数器

共 12 字节

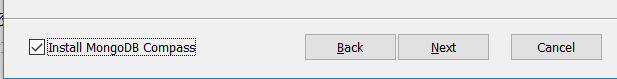
# Mongdodb的安装

1. 下载地址

<https://www.mongodb.com/download-center?jmp=nav#community>

本文以mongodb 3.6为示例

1. 安装mongdo到指定的目录，如：D:\MongoDB\Server\3.6
2. 安装过程中，不要勾选 Install MongoDB Compass， 安装官方的可视化工具



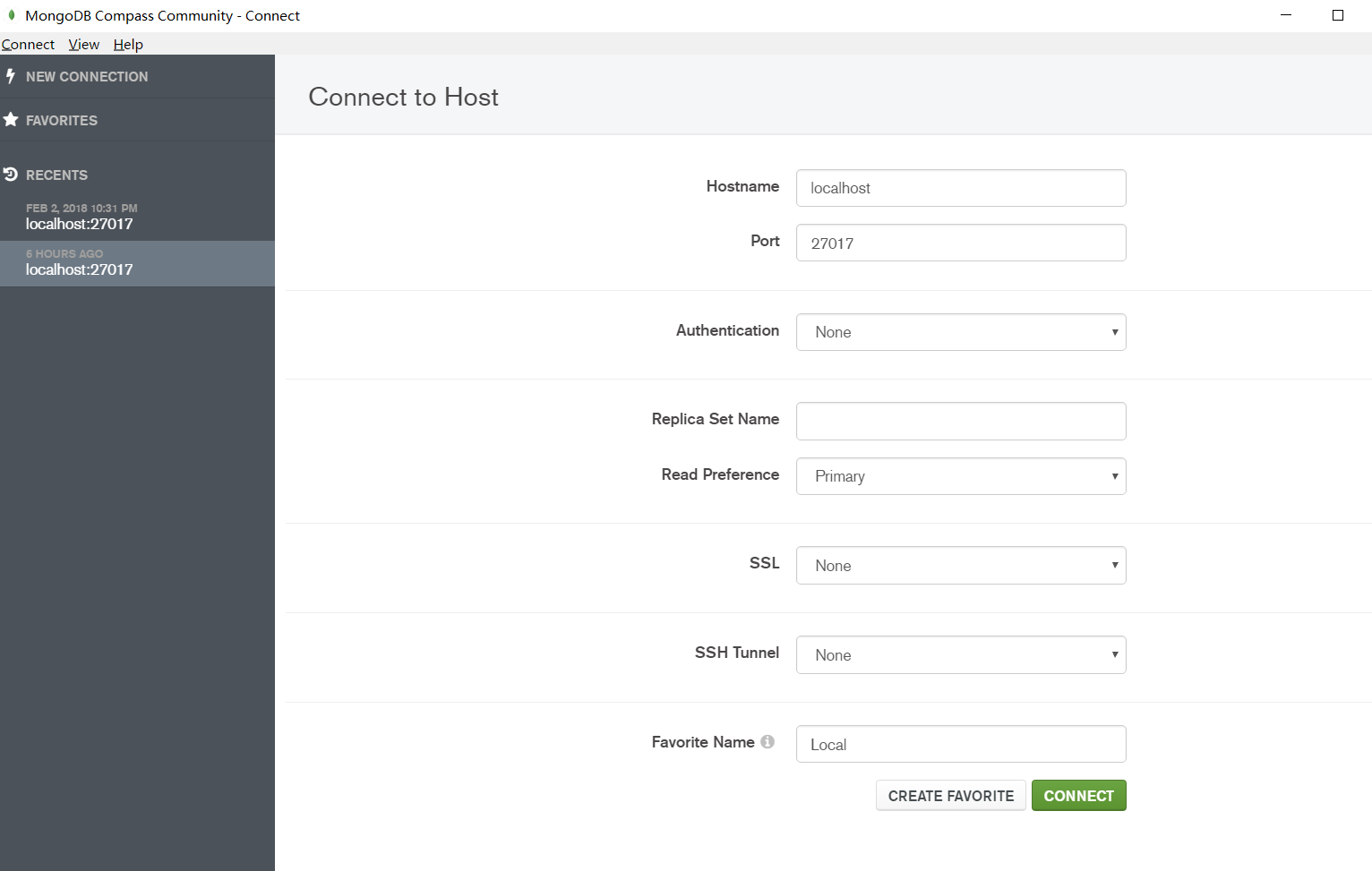
1. 将 D:\MongoDB\Server\3.6\bin 添加到环境变量中
2. 配置mongod.conf文件(如下)，并且在D:\MongoDB下新建 data和log 文件夹

|  |
| --- |
| #数据库数据存放目录  dbpath=D:\MongoDB\data  #数据库日志存放目录  logpath=D:\MongoDB\log\mongodb.log  #以追加的方式记录日志  logappend = true  #端口号 默认为27017  port=27017  #开启用户认证  auth=true  #mongodb所绑定的ip地址，绑定后只能通过127访问  bind\_ip = 127.0.0.1  #启用日志文件，默认启用  journal=true  #这个选项可以过滤掉一些无用的日志信息，若需要调试使用请设置为false  quiet=true  #不允许全表扫描  notablescan=false |

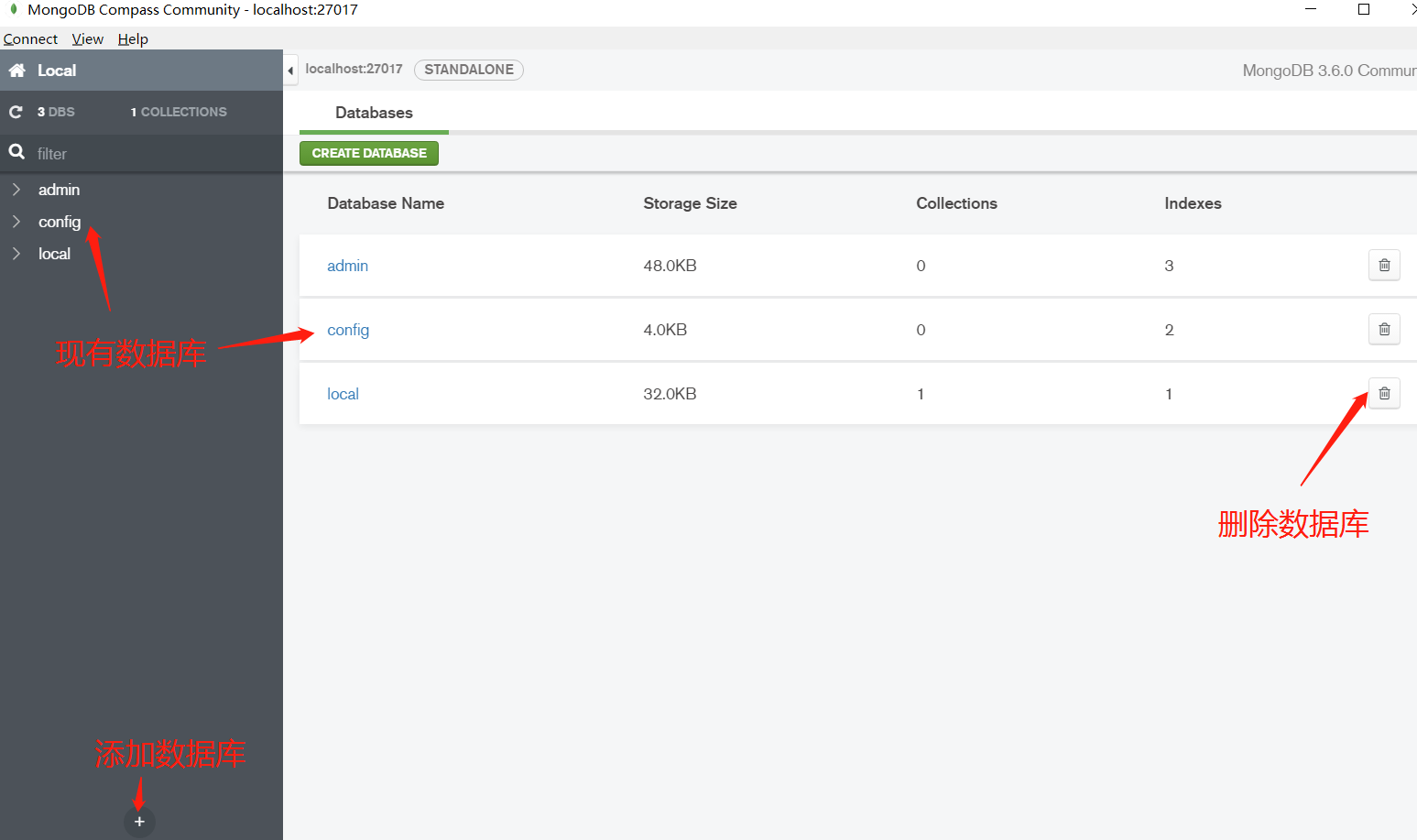
1. 以管理员身份打开cmd命令窗口，切换到 D:\MongoDB\Server\3.6\bin 目录下
2. 将mongodb安装为服务，在cmd命令窗口中输入 mongod --install -f “D:\MongoDB\Server\3.6\mongod.conf”
3. 安装完成后，使用 net start mongodb 启动服务， net stop mongodb 停止服务
4. 官方手册地址： <https://docs.mongodb.com/manual/>
5. 在mongodb中，table叫做collection

# MongoDB Compass Community

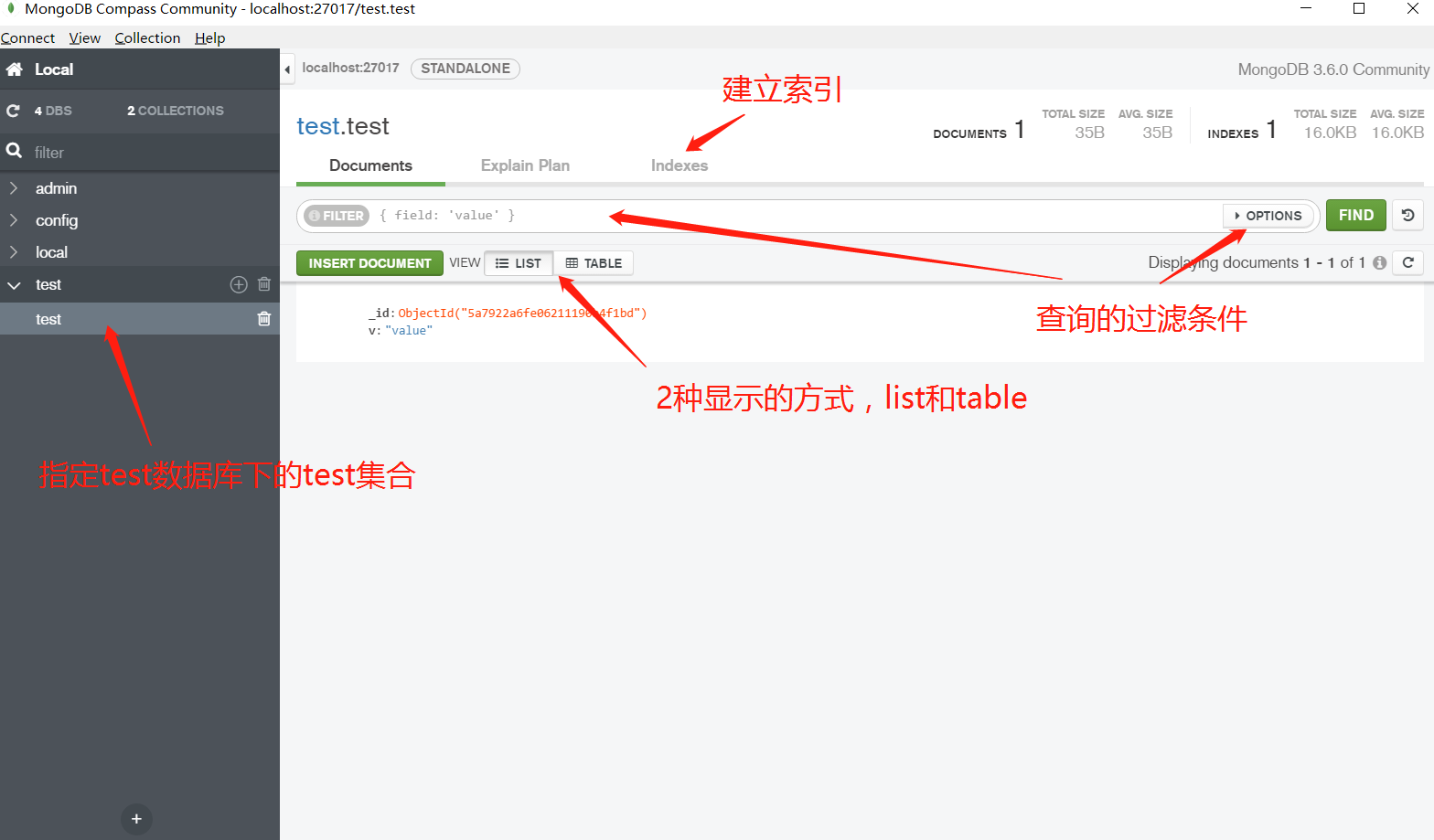
1. 工具使用帮助 https://docs.mongodb.com/compass/current/
2. 打开程序，填写对应的hostname，port等，点击 connnect 连接



1. 选择一个现有数据库或新建一个数据库



1. 建立数据库之后，进入数据库



# MongoDB命令

## 基础命令

|  |
| --- |
| /\*  连接mongodb  如果设置了环境变量，那么在cmd命令窗口就可以直接执行，否则需要切换到mongodb的bin目录  \*/  或  mongo |
| mongo 远程主机ip或DNS:MongoDB端口号/数据库名 -u user -p password  /\*  查看命令的帮助  \*/  Help  /\*  显示所有数据库  \*/ |
| show dbs  /\*  使用某个数据库，如果没有则新建  \*/ |
| use db\_name  /\*  显示所有集合  \*/  show collections  /\*  删除指定集合  \*/  db.collection\_name.drop()  /\*  删除指定数据库，需要先切换到需要删除的数据库下use db\_name  \*/  db.dropDatabase() |

## 用户命令

|  |
| --- |
| /\*  新建用户，admin必须切换到admin数据库：use admin  \*/  或  db.createUser({user:"admin",pwd:" admin", roles:[{role:"root", db:"admin"}]}) |
| /\*  使用用户登录，config中有个配置项auth=true  \*/  mongo  use admin  db.auth('admin', 'qwe123')  /\*  修改密码  \*/  db.changeUserPassword("admin", "admin123")  /\*  删除用户  \*/  db.dropUser("user\_name")  /\*  显示用户的详情  \*/  db.getUser("user\_name")  /\*  显示所有权限  \*/ |
| show roles  /\*  创建权限  \*/  use admin |
| db.createRole({role:"test\_role", privileges:[{resource:{db:"", collection:""},actions: [ "serverStatus" ] }], roles:[]})  /\*0  删除权限  \*/  db.dropRole("test\_role") |

## 增加数据

插入数据时，当没有这个集合时，会自动创建一个新的collection，没有写入 \_id时，会自动创建一个 \_id 的字段

|  |
| --- |
| /\*  db.collection.insert(  <document or array of documents>,  {  writeConcern: <document>,  ordered: <boolean>  }  writeConcern：{ w: <value>, j: <boolean>, wtimeout: <number> }  w：1：应答式写入，确认已经写入单个节点，默认就是这个  0：非应答式，不返回任何东西，不知道是否写入成功  >1：该值用于设定写入节点的数目，包括主节点  majority：大多数，适用于集群架构，要求写入操作已经传递到绝大多数投票节点  <tag set>：要求写入操作已经传递到指定tag标记副本集中的成员后进行应答  j：该选项要求确认写操作已经写入journal日志之后应答客户端(需要开启journal功能)  wtimeout：单位毫秒，超时时间，防止卡死  ordered：为真，顺序插入，发生错误后，不处理剩余的数据，直接返回，为假，无序插入，发生错误，继续处理剩余的部分，默认为 true  )\*/  db.products.insert(  [  { \_id: 11, item: "pencil", qty: 50, type: "no.2" },  { item: "pen", qty: 20 },  { item: "eraser", qty: 25 }  ]  ) |
| db.products.insert(  { item: "envelopes", qty : 100, type: "Clasp" },  { writeConcern: { w: 1,j:true, wtimeout: 5000 } }  )  /\*  db.collection.insertOne(  <document>,  {  writeConcern: <document>  }  )  \*/  db.products.insertOne( { item: "card", qty: 15 } );  /\*  db.collection.insertMany(  [ <document 1> , <document 2>, ... ],  {  writeConcern: <document>,  ordered: <boolean>  }  )  \*/  db.products.insertMany( [  { item: "card", qty: 15 },  { item: "envelope", qty: 20 },  { item: "stamps" , qty: 30 }  ] ); |

## 查看数据

|  |
| --- |
| /\*  db.collection.find(query, projection)  query：查询条件  projection： { field1: <value>, field2: <value> ... } ，value为1或0，返回该字段或者不返回，\_id 如果不指定 \_id:0 ，那么会返回  \*/  db.products.find({qty:20})  db.products.find({qty:20}, {\_id:0, qty:1}) |
| /\*  联合查询 $lookup  \*/  db.orders.insert([  { "\_id" : 1, "item" : "abc", "price" : 12, "quantity" : 2 },  { "\_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1 },  { "\_id" : 3 }  ])  db.inventory.insert([  { "\_id" : 1, "sku" : "abc", description: "product 1", "instock" : 120 },  { "\_id" : 2, "sku" : "def", description: "product 2", "instock" : 80 },  { "\_id" : 3, "sku" : "ijk", description: "product 3", "instock" : 60 },  { "\_id" : 4, "sku" : "jkl", description: "product 4", "instock" : 70 },  { "\_id" : 5, "sku": null, description: "Incomplete" },  { "\_id" : 6 }  ])  db.orders.aggregate([  {  $lookup:  {  from: "inventory",  localField: "item",  foreignField: "sku",  as: "inventory\_docs"  }  }  ]) |

## 修改数据

|  |
| --- |
| db.inventory.insertMany( [  { item: "canvas", qty: 100, size: { h: 28, w: 35.5, uom: "cm" }, status: "A" },  { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },  { item: "mat", qty: 85, size: { h: 27.9, w: 35.5, uom: "cm" }, status: "A" },  { item: "mousepad", qty: 25, size: { h: 19, w: 22.85, uom: "cm" }, status: "P" },  { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "P" },  { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },  { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },  { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" },  { item: "sketchbook", qty: 80, size: { h: 14, w: 21, uom: "cm" }, status: "A" },  { item: "sketch pad", qty: 95, size: { h: 22.85, w: 30.5, uom: "cm" }, status: "A" }  ] );  db.collection.updateOne(<filter>, <update>, <options>)  db.collection.updateMany(<filter>, <update>, <options>)  db.collection.replaceOne(<filter>, <replacement>, <options>)  options:  upsert： 如果有则修改，否则新增，默认为true  w: 同上面的w  wtimeout：超时时间  j：是否写入日志  db.inventory.updateOne(  { item: "paper"},  {  $set:{'qty':101}  }  )  db.inventory.updateOne(  { item: "paper" },  {  $set: {"size.uom": "cm", status: "P" },  $currentDate: { lastModified: true }  }  )  # update inventory set ...... where qty < 50  db.inventory.updateMany(  { "qty": { $lt: 50 } },  {  $set: { "size.uom": "in", status: "P" }  }  )  db.inventory.replaceOne(  { item: "paper" },  { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 40 } ] }  ) |
|  |

## 删除数据

|  |
| --- |
| db.inventory.insertMany( [  { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },  { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "P" },  { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },  { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },  { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" },  ] );  db.collection.deleteMany()  db.collection.deleteOne()  # 删除所有，必须有 {}  db.inventory.deleteMany({})  db.inventory.deleteMany({ status : "A" })  db.inventory.deleteOne( { status: "D" } ) |

## 索引

1. 索引基础：

MongoDB的索引几乎与传统的关系型数据库一模一样，这其中也包括一些基本的优化技巧。下面是创建索引的命令：

> db.test.createIndex({"username":1})

可以通过下面的名称查看索引是否已经成功建立：

> db.test.getIndexes()

删除索引的命令是：

> db.test.dropIndex({"username":1})

在MongoDB中，我们同样可以创建复合索引，如：

-- 数字1表示username键的索引按升序存储，-1表示age键的索引按照降序方式存储。

> db.test.createIndex({"username":1, "age":-1})

> db.test.createIndex({"age":-1, "username":1})

该索引被创建后，基于username和age的查询将会用到该索引，或者是基于username的查询也会用到该索引，但是只是基于age的查询将不会用到该复合索引。因此可以说，如果想用到复合索引，必须在查询条件中包含复合索引中的前N个索引列。然而如果查询条件中的键值顺序和复合索引中的创建顺序不一致的话，MongoDB可以智能的帮助我们调整该顺序，以便使复合索引可以为查询所用。如：

> db.test.find({"age": 30, "username": "stephen"})

对于上面示例中的查询条件，MongoDB在检索之前将会动态的调整查询条件文档的顺序，以使该查询可以用到刚刚创建的复合索引。

我们可以为内嵌文档创建索引，其规则和普通文档没有任何差别，如：

> db.test.createIndex({"comments.date":1})

对于上面创建的索引，MongoDB都会根据索引的keyname和索引方向为新创建的索引自动分配一个索引名，下面的命令可以在创建索引时为其指定索引名，如：

> db.test.createIndex({"username":1},{"name":"testindex"})

随着集合的增长，需要针对查询中大量的排序做索引。如果没有对索引的键调用sort，MongoDB需要将所有数据提取到内存并排序。因此在做无索引排序时，如果数据量过大以致无法在内存中进行排序，此时MongoDB将会报错。

1. 唯一索引：

在缺省情况下创建的索引均不是唯一索引。下面的示例将创建唯一索引，如：

> db.test.createIndex({"userid":1},{"unique":true})

如果再次插入userid重复的文档时，MongoDB将报错，以提示插入重复键，如：

> db.test.insert({"userid":5})

> db.test.insert({"userid":5})

E11000 duplicate key error index: test.test.$userid\_1 dup key: { : 5.0 }

如果插入的文档中不包含userid键，那么该文档中该键的值为null，如果多次插入类似的文档，MongoDB将会报出同样的错误，如：

> db.test.insert({"userid1":5})

> db.test.insert({"userid1":5})

E11000 duplicate key error index: test.test.$userid\_1 dup key: { : null }

如果在创建唯一索引时已经存在了重复项，我们可以通过下面的命令帮助我们在创建唯一索引时消除重复文档，仅保留发现的第一个文档，如：

--先删除刚刚创建的唯一索引。

> db.test.dropIndex({"userid":1})

--插入测试数据，以保证集合中有重复键存在。

> db.test.remove()

> db.test.insert({"userid":5})

> db.test.insert({"userid":5})

我们同样可以创建复合唯一索引，即保证复合键值唯一即可。如：

> db.test.createIndex({"userid":1,"age":1},{"unique":true})

1. 使用explain：

explain是非常有用的工具，会帮助你获得查询方面诸多有用的信息。只要对游标调用该方法，就可以得到查询细节。explain会返回一个文档，而不是游标本身。如：

> db.orders.find({\_id:1}).explain()

|  |
| --- |
| {  "queryPlanner" : {  "plannerVersion" : 1,  "namespace" : "test.orders",  "indexFilterSet" : false,  "parsedQuery" : {  "\_id" : {  "$eq" : 1  }  },  "winningPlan" : {  "stage" : "IDHACK"  },  "rejectedPlans" : [ ]  },  "serverInfo" : {  "host" : "Terry",  "port" : 27017,  "version" : "3.6.0",  "gitVersion" : "a57d8e71e6998a2d0afde7edc11bd23e5661c915"  },  "ok" : 1  } |

explain会返回查询使用的索引情况，耗时和扫描文档数的统计信息。

1. 索引管理：

如果在为已有数据的文档创建索引时，可以执行下面的命令，以使MongoDB在后台创建索引，这样的创建时就不会阻塞其他操作。但是相比而言，以阻塞方式创建索引，会使整个创建过程效率更高，但是在创建时MongoDB将无法接收其他的操作。

> db.test.createIndex({"username":1},{"background":true})

应用停止使用，数据库可以锁定的情况下不要添加background 参数

应用运行时，添加索引，必须使用background参数

# 常见问题

1. MongoDB 数据库级锁：MongoDB 只能提供 库级粒度锁，尽量避免长时间占用写锁操作，如果有一些集合操作实在难以避免，可以考虑把这个集合放到一个单独的 MongoDB 库里，因为 MongoDB 不同库锁是相互隔离的，分离集合可以避免某一个集合操作引发全局阻塞问题。
2. 建索引：为了避免上面第一个问题的出现，使用backgroud，db.collection\_name.createIndex({field\_name: 1}, {background: 1})
3. 尽量避免使用embed document：document中嵌套document，尽量避免，推荐使用普通的类似关系数据库的外键关联