# MongoDB 概念解析

在mongodb中基本的概念是文档、集合、数据库

下表将帮助您更容易理解Mongo中的一些概念：

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

通过下图实例，我们也可以更直观的的了解Mongo中的一些概念：



## 数据库（Database）

一个mongodb中可以建立多个数据库。

MongoDB的默认数据库为"db"，该数据库存储在data目录中。

MongoDB的单个实例可以容纳多个独立的数据库，每一个都有自己的集合和权限，不同的数据库也放置在不同的文件中。

**"show dbs"** 命令可以显示所有数据的列表。

执行 **"db"** 命令可以显示当前数据库对象或集合。

运行**"use"**命令，可以连接到一个指定的数据库。

数据库也通过名字来标识。数据库名可以是满足以下条件的任意UTF-8字符串。

* 不能是空字符串（"")。
* 不得含有' '（空格)、.、$、/、\和\0 (空宇符)。
* 应全部小写。
* 最多64字节。

有一些数据库名是保留的，可以直接访问这些有特殊作用的数据库。

* **admin**： 从权限的角度来看，这是"root"数据库。要是将一个用户添加到这个数据库，这个用户自动继承所有数据库的权限。一些特定的服务器端命令也只能从这个数据库运行，比如列出所有的数据库或者关闭服务器。
* **local:** 这个数据永远不会被复制，可以用来存储限于本地单台服务器的任意集合
* **config**: 当Mongo用于分片设置时，config数据库在内部使用，用于保存分片的相关信息。

## 文档(document，相当于row)

文档是一个键值(key-value)对(即BSON)。MongoDB 的文档不需要设置相同的字段，并且相同的字段不需要相同的数据类型，这与关系型数据库有很大的区别，也是 MongoDB 非常突出的特点。

一个简单的文档例子如下：

{"site":"www.runoob.com", "name":"菜鸟教程"}

下表列出了 RDBMS 与 MongoDB 对应的术语：

|  |  |
| --- | --- |
| **RDBMS** | **MongoDB** |
| 数据库 | 数据库 |
| 表格 | 集合 |
| 行 | 文档 |
| 列 | 字段 |
| 表联合 | 嵌入文档 |
| 主键 | 主键 (MongoDB 提供了 key 为 \_id ) |
| **数据库服务和客户端** | |
| Mysqld/Oracle | mongod |
| mysql/sqlplus | mongo |

需要注意的是：

1. 文档中的键/值对是有序的。
2. 文档中的值不仅可以是在双引号里面的字符串，还可以是其他几种数据类型（甚至可以是整个嵌入的文档)。
3. MongoDB区分类型和大小写。
4. MongoDB的文档不能有重复的键。
5. 文档的键是字符串。除了少数例外情况，键可以使用任意UTF-8字符。

文档键命名规范：

* 键不能含有\0 (空字符)。这个字符用来表示键的结尾。
* .和$有特别的意义，只有在特定环境下才能使用。
* 以下划线"\_"开头的键是保留的(不是严格要求的)。

## 集合（collection，相当于table）

集合就是 MongoDB 文档组，类似于 RDBMS （关系数据库管理系统：Relational Database Management System)中的表格。

集合存在于数据库中，集合没有固定的结构，这意味着你在对集合可以插入不同格式和类型的数据，但通常情况下我们插入集合的数据都会有一定的关联性。

{"site":"www.baidu.com"}

{"site":"www.google.com","name":"Google"}

{"site":"www.runoob.com","name":"菜鸟教程","num":5}

当第一个文档插入时，集合就会被创建。

合法的集合名：

* 集合名不能是空字符串""。
* 集合名不能含有\0字符（空字符)，这个字符表示集合名的结尾。
* 集合名不能以"system."开头，这是为系统集合保留的前缀。
* 用户创建的集合名字不能含有保留字符。有些驱动程序的确支持在集合名里面包含，这是因为某些系统生成的集合中包含该字符。除非你要访问这种系统创建的集合，否则千万不要在名字里出现$。

### capped collections

Capped collections 就是固定大小的collection。

它有很高的性能以及队列过期的特性(过期按照插入的顺序). 有点和 "RRD" 概念类似。

Capped collections是高性能自动的维护对象的插入顺序。它非常适合类似记录日志的功能 和标准的collection不同，你必须要显式的创建一个capped collection， 指定一个collection的大小，单位是字节。collection的数据存储空间值提前分配的。

要注意的是指定的存储大小包含了数据库的头信息。

db.createCollection("mycoll", {capped:true, size:100000})

* 在capped collection中，你能添加新的对象。
* 能进行更新，然而，对象不会增加存储空间。如果增加，更新就会失败 。
* 数据库不允许进行删除。使用drop()方法删除collection所有的行。
* 注意: 删除之后，你必须显式的重新创建这个collection。
* 在32bit机器中，capped collection最大存储为1e9( 1X109)个字节。

## 元数据

数据库的信息是存储在集合中。它们使用了系统的命名空间：

dbname.system.\*

在MongoDB数据库中名字空间 <dbname>.system.\* 是包含多种系统信息的特殊集合(Collection)，如下:

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

对于修改系统集合中的对象有如下限制。

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

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

## MongoDB 数据类型

下表为MongoDB中常用的几种数据类型。

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| 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。 |

# MongoDB - 连接

## 启动 MongoDB服务

只需要在MongoDB安装目录的bin目录下执行'mongod'即可。

执行启动操作后，mongodb在输出一些必要信息后不会输出任何信息，之后就等待连接的建立，当连接被建立后，就会开始打印日志信息。

## 通过shell连接MongoDB服务

你可以通过执行以下命令来连接MongoDB的服务。

mongodb://localhost

**注意：**localhost为主机名，这个选项是必须的：

当你执行以上命令时，你可以看到以下输出结果：

$ ./mongo

MongoDB shell version: 3.0.6

connecting to: test

> mongodb://localhostmongodb://localhost

...

## MongoDB连接命令格式

使用用户名和密码连接到MongoDB服务器，你必须使用 'username:password@hostname/dbname' 格式，'username'为用户名，'password' 为密码。

使用用户名和密码连接登陆到默认数据库：

mongodb://admin:123456@localhost/test

## 更多连接实例

连接本地数据库服务器，端口是默认的。

mongodb://localhost

使用用户名fred，密码foobar登录localhost的admin数据库。

mongodb://fred:foobar@localhost

使用用户名fred，密码foobar登录localhost的baz数据库。

mongodb://fred:foobar@localhost/baz

连接 replica pair, 服务器1为example1.com服务器2为example2。

mongodb://example1.com:27017,example2.com:27017

连接 replica set 三台服务器 (端口 27017, 27018, 和27019):

mongodb://localhost,localhost:27018,localhost:27019

连接 replica set 三台服务器, 写入操作应用在主服务器 并且分布查询到从服务器。

mongodb://host1,host2,host3/?slaveOk=true

直接连接第一个服务器，无论是replica set一部分或者主服务器或者从服务器。

mongodb://host1,host2,host3/?connect=direct;slaveOk=true

当你的连接服务器有优先级，还需要列出所有服务器，你可以使用上述连接方式。

安全模式连接到localhost:

mongodb://localhost/?safe=true

以安全模式连接到replica set，并且等待至少两个复制服务器成功写入，超时时间设置为2秒。

mongodb://host1,host2,host3/?safe=true;w=2;wtimeoutMS=2000

## 参数选项说明

标准格式：

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

标准的连接格式包含了多个选项(options)，如下所示：

|  |  |
| --- | --- |
| **选项** | **描述** |
| replicaSet=name | 验证replica set的名称。 Impliesconnect=replicaSet. |
| slaveOk=true|false | * true:在connect=direct模式下，驱动会连接第一台机器，即使这台服务器不是主。在connect=replicaSet模式下，驱动会发送所有的写请求到主并且把读取操作分布在其他从服务器。 * false: 在 connect=direct模式下，驱动会自动找寻主服务器. 在connect=replicaSet 模式下，驱动仅仅连接主服务器，并且所有的读写命令都连接到主服务器。 |
| safe=true|false | * + true: 在执行更新操作之后，驱动都会发送getLastError命令来确保更新成功。(还要参考 wtimeoutMS).   false: 在每次更新之后，驱动不会发送getLastError来确保更新成功。 |
| w=n | 驱动添加 { w : n } 到getLastError命令. 应用于safe=true。 |
| wtimeoutMS=ms | 驱动添加 { wtimeout : ms } 到 getlasterror 命令. 应用于 safe=true. |
| fsync=true|false | * true: 驱动添加 { fsync : true } 到 getlasterror 命令.应用于 safe=true. * false: 驱动不会添加到getLastError命令中。 |
| journal=true|false | 如果设置为 true, 同步到 journal (在提交到数据库前写入到实体中). 应用于 safe=true |
| connectTimeoutMS=ms | 可以打开连接的时间。 |
| socketTimeoutMS=ms | 发送和接受sockets的时间。 |

# MongoDB 创建数据库

## 语法

MongoDB 创建数据库的语法格式如下：

use DATABASE\_NAME

如果数据库不存在，则创建数据库，否则切换到指定数据库。

## 实例

以下实例我们创建了数据库 runoob:

> use runoob

switched to db runoob

> db

runoob

>

# MongoDB 删除数据库

## 语法

MongoDB 删除数据库的语法格式如下：

db.dropDatabase()

删除当前数据库，默认为 test，你可以使用 db 命令查看当前数据库名。

## 实例

切换到数据库 runoob：

> use runoob

switched to db runoob

>

执行删除命令：

> db.dropDatabase()

{ "dropped" : "runoob", "ok" : 1 }

最后，我们再通过 show dbs 命令数据库是否删除成功：

> show dbs

local 0.078GB

test 0.078GB

>

# MongoDB 文档操作

## 插入文档

MongoDB 使用 insert() 或 save() 方法向集合中插入文档，语法如下：

db.COLLECTION\_NAME.insert(document)

### 实例

以下文档可以存储在 MongoDB 的 runoob 数据库 的 col集合中：

>db.col.insert({title: 'MongoDB 教程',

description: 'MongoDB 是一个 Nosql 数据库',

by: '菜鸟教程',

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

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

likes: 100

})

以上实例中 col 是我们的集合名，前一章节我们已经创建过了，如果该集合不在该数据库中， MongoDB 会自动创建该集合并插入文档。

查看已插入文档：

> db.col.find()

{ "\_id" : ObjectId("56064886ade2f21f36b03134"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 100 }

>

我们也可以将数据定义为一个变量，如下所示：

> document=({title: 'MongoDB 教程',

description: 'MongoDB 是一个 Nosql 数据库',

by: '菜鸟教程',

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

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

likes: 100

});

执行后显示结果如下：

{

"title" : "MongoDB 教程",

"description" : "MongoDB 是一个 Nosql 数据库",

"by" : "菜鸟教程",

"url" : "http://www.runoob.com",

"tags" : [

"mongodb",

"database",

"NoSQL"

],

"likes" : 100

}

执行插入操作：

> db.col.insert(document)

WriteResult({ "nInserted" : 1 })

>

插入文档你也可以使用 db.col.save(document) 命令。如果不指定 \_id 字段 save() 方法类似于 insert() 方法。如果指定 \_id 字段，则会更新该 \_id 的数据。

## 更新文档

MongoDB 使用 **update()** 和 **save()** 方法来更新集合中的文档。接下来让我们详细来看下两个函数的应用及其区别。

**update() 方法**

update() 方法用于更新已存在的文档。语法格式如下：

db.collection.update(

<query>,

<update>,

{

upsert: <boolean>,

multi: <boolean>,

writeConcern: <document>

}

)

**参数说明：**

* **query**: update的查询条件，类似sql update查询内where后面的。
* **update**: update的对象和一些更新的操作符（如$,$inc...）等，也可以理解为sql update查询内set后面的
* **upsert**: 可选，这个参数的意思是，如果不存在update的记录，是否插入objNew,true为插入，默认是false，不插入。
* **multi**: 可选，mongodb 默认是false,只更新找到的第一条记录，如果这个参数为true,就把按条件查出来多条记录全部更新。
* **writeConcern**:可选，抛出异常的级别。

### 实例

我们在集合 col 中插入如下数据：

>db.col.insert({

title: 'MongoDB 教程',

description: 'MongoDB 是一个 Nosql 数据库',

by: '菜鸟教程',

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

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

likes: 100

})

接着我们通过 update() 方法来更新标题(title):

>db.col.update({'title':'MongoDB 教程'},{$set:{'title':'MongoDB'}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 }) # 输出信息

> db.col.find().pretty()

{

"\_id" : ObjectId("56064f89ade2f21f36b03136"),

"title" : "MongoDB",

"description" : "MongoDB 是一个 Nosql 数据库",

"by" : "菜鸟教程",

"url" : "http://www.runoob.com",

"tags" : [

"mongodb",

"database",

"NoSQL"

],

"likes" : 100

}

>

可以看到标题(title)由原来的 "MongoDB 教程" 更新为了 "MongoDB"。

以上语句只会修改第一条发现的文档，如果你要修改多条相同的文档，则需要设置 multi 参数为 true。

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

**Save方法（）**

save() 方法通过传入的文档来替换已有文档。语法格式如下：

db.collection.save(

<document>,

{

writeConcern: <document>

}

)

**参数说明：**

* **document**: 文档数据。
* **writeConcern**:可选，抛出异常的级别。

### 实例

以下实例中我们替换了 \_id 为 56064f89ade2f21f36b03136 的文档数据：

>db.col.save({

"\_id" : ObjectId("56064f89ade2f21f36b03136"),

"title" : "MongoDB",

"description" : "MongoDB 是一个 Nosql 数据库",

"by" : "Runoob",

"url" : "http://www.runoob.com",

"tags" : [

"mongodb",

"NoSQL"

],

"likes" : 110

})

### 更多实例

只更新第一条记录：

db.col.update( { "count" : { $gt : 1 } } , { $set : { "test2" : "OK"} } );

全部更新：

db.col.update( { "count" : { $gt : 3 } } , { $set : { "test2" : "OK"} },false,true );

只添加第一条：

db.col.update( { "count" : { $gt : 4 } } , { $set : { "test5" : "OK"} },true,false );

全部添加加进去:

db.col.update( { "count" : { $gt : 5 } } , { $set : { "test5" : "OK"} },true,true );

全部更新：

db.col.update( { "count" : { $gt : 15 } } , { $inc : { "count" : 1} },false,true );

只更新第一条记录：

db.col.update( { "count" : { $gt : 10 } } , { $inc : { "count" : 1} },false,false );

## 删除文档

MongoDB remove()函数是用来移除集合中的数据。

remove() 方法的基本语法格式如下所示：

db.collection.remove(

<query>,

<justOne>

)

2.6 版本以后的MongoDB ，语法格式如下：

db.collection.remove(

<query>,

{

justOne: <boolean>,

writeConcern: <document>

}

)

**参数说明：**

* **query**:（可选）删除的文档的条件。
* **justOne**: （可选）如果设为 true 或 1，则只删除一个文档。
* **writeConcern**:（可选）抛出异常的级别。

### 实例

我们移除 title 为 'MongoDB 教程' 的文档：

>db.col.remove({'title':'MongoDB 教程'})

如果你只想删除第一条找到的记录可以设置 justOne 为 1，如下所示：

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

如果你想删除所有数据，可以使用以下方式（类似常规 SQL 的 truncate 命令）：

>db.col.remove({})

>db.col.find()

>

## 查询文档

MongoDB 查询数据的语法格式如下：

>db.COLLECTION\_NAME.find()

find() 方法以非结构化的方式来显示所有文档。

如果你需要以易读的方式来读取数据，可以使用 pretty() 方法，语法格式如下：

>db.col.find().pretty()

pretty() 方法以格式化的方式来显示所有文档。

### 实例

以下实例我们查询数据库runoob:

> db.col.find().pretty()

{

"\_id" : ObjectId("56063f17ade2f21f36b03133"),

"title" : "MongoDB 教程",

"description" : "MongoDB 是一个 Nosql 数据库",

"by" : "菜鸟教程",

"url" : "http://www.runoob.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 与 RDBMS Where 语句比较

如果你熟悉常规的 SQL 数据，通过下表可以更好的理解 MongoDB 的条件语句查询：

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

### MongoDB AND 条件

MongoDB 的 find() 方法可以传入多个键(key)，每个键(key)以逗号隔开，及常规 SQL 的 AND 条件。

语法格式如下：

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

以下实例通过 **by** 和 **title** 键来查询 **菜鸟教程** 中 **MongoDB 教程** 的数据

> db.col.find({"by":"菜鸟教程", "title":"MongoDB 教程"}).pretty()

以上实例中类似于 WHERE 语句：**WHERE by='菜鸟教程' AND title='MongoDB 教程'**

### MongoDB OR 条件

MongoDB OR 条件语句使用了关键字 **$or**,语法格式如下：

>db.col.find(

{

$or: [

{key1: value1}, {key2:value2}

]

}

).pretty()

以下实例中，我们演示了查询键 **by** 值为 菜鸟教程 或键 **title** 值为 **MongoDB 教程** 的文档。

>db.col.find({$or:[{"by":"菜鸟教程"},{"title": "MongoDB 教程"}]}).pretty()

### AND 和 OR 联合使用

以下实例演示了 AND 和 OR 联合使用，类似常规 SQL 语句为： **'where likes>50 AND (by = '菜鸟教程' OR title = 'MongoDB 教程')'**

>db.col.find({"likes": {$gt:50}, $or: [{"by": "菜鸟教程"},{"title": "MongoDB 教程"}]}).pretty()

# MongoDB 条件操作符

条件操作符用于比较两个表达式并从mongoDB集合中获取数据。

MongoDB中条件操作符有：

* (>) 大于 - $gt
* (<) 小于 - $lt
* (>=) 大于等于 - $gte
* (<= ) 小于等于 - $lte

## MongoDB (>) 大于操作符 - $gt

如果你想获取 "col" 集合中 "likes" 大于 100 的数据，你可以使用以下命令：

db.col.find({"likes" : {$gt : 100}})

类似于SQL语句：

Select \* from col where likes > 100;

## MongoDB（>=）大于等于操作符 - $gte

如果你想获取"col"集合中 "likes" 大于等于 100 的数据，你可以使用以下命令：

db.col.find({likes : {$gte : 100}})

类似于SQL语句：

Select \* from col where likes >=100;

## MongoDB (<) 小于操作符 - $lt

如果你想获取"col"集合中 "likes" 小于 150 的数据，你可以使用以下命令：

db.col.find({likes : {$lt : 150}})

类似于SQL语句：

Select \* from col where likes < 150;

## MongoDB (<=) 小于操作符 - $lte

如果你想获取"col"集合中 "likes" 小于等于 150 的数据，你可以使用以下命令：

db.col.find({likes : {$lte : 150}})

类似于SQL语句：

Select \* from col where likes <= 150;

## MongoDB 使用 (<) 和 (>) 查询 - $lt 和 $gt

如果你想获取"col"集合中 "likes" 大于100，小于 200 的数据，你可以使用以下命令：

db.col.find({likes : {$lt :200, $gt : 100}})

类似于SQL语句：

Select \* from col where likes>100 AND likes<200;

# MongoDB $type 操作符

$type操作符是基于BSON类型来检索集合中匹配的数据类型，并返回结果。

MongoDB 中可以使用的类型如下表所示：

|  |  |  |
| --- | --- | --- |
| **类型** | **数字** | **备注** |
| Double | 1 |  |
| String | 2 |  |
| Object | 3 |  |
| Array | 4 |  |
| Binary data | 5 |  |
| Undefined | 6 | 已废弃。 |
| Object id | 7 |  |
| Boolean | 8 |  |
| Date | 9 |  |
| Null | 10 |  |
| Regular Expression | 11 |  |
| JavaScript | 13 |  |
| Symbol | 14 |  |
| JavaScript (with scope) | 15 |  |
| 32-bit integer | 16 |  |
| Timestamp | 17 |  |
| 64-bit integer | 18 |  |
| Min key | 255 | Query with -1. |
| Max key | 127 |  |

如果想获取 "col" 集合中 title 为 String 的数据，你可以使用以下命令：

db.col.find({"title" : {$type : 2}})

# MongoDB Limit与Skip方法

## MongoDB Limit() 方法

如果你需要在MongoDB中读取指定数量的数据记录，可以使用MongoDB的Limit方法，limit()方法接受一个数字参数，该参数指定从MongoDB中读取的记录条数。

limit()方法基本语法如下所示：

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

## MongoDB Skip() 方法

我们除了可以使用limit()方法来读取指定数量的数据外，还可以使用skip()方法来跳过指定数量的数据，skip方法同样接受一个数字参数作为跳过的记录条数。

skip() 方法脚本语法格式如下：

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

# MongoDB 排序

## MongoDB sort()方法

在MongoDB中使用使用sort()方法对数据进行排序，sort()方法可以通过参数指定排序的字段，并使用 1 和 -1 来指定排序的方式，其中 1 为升序排列，而-1是用于降序排列。

sort()方法基本语法如下所示：

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

**注：** 如果没有指定sort()方法的排序方式，默认按照文档的升序排列。

# MongoDB 索引

索引通常能够极大的提高查询的效率，如果没有索引，MongoDB在读取数据时必须扫描集合中的每个文件并选取那些符合查询条件的记录。

这种扫描全集合的查询效率是非常低的，特别在处理大量的数据时，查询可以要花费几十秒甚至几分钟，这对网站的性能是非常致命的。

索引是特殊的数据结构，索引存储在一个易于遍历读取的数据集合中，索引是对数据库表中一列或多列的值进行排序的一种结构

## ensureIndex() 方法

MongoDB使用 ensureIndex() 方法来创建索引。

ensureIndex()方法基本语法格式如下所示：

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

语法中 Key 值为你要创建的索引字段，1为指定按升序创建索引，如果你想按降序来创建索引指定为-1即可。

ensureIndex() 方法中你也可以设置使用多个字段创建索引（关系型数据库中称作复合索引）。

ensureIndex() 接收可选参数，可选参数列表如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Type** | | **Description** |
| background | | Boolean | 建索引过程会阻塞其它数据库操作，background可指定以后台方式创建索引，即增加 "background" 可选参数。 "background" 默认值为**false**。 |
| unique | | Boolean | 建立的索引是否唯一。指定为true创建唯一索引。默认值为**false**. |
| name | | string | 索引的名称。如果未指定，MongoDB的通过连接索引的字段名和排序顺序生成一个索引名称。 |
| dropDups | | Boolean | 在建立唯一索引时是否删除重复记录,指定 true 创建唯一索引。默认值为 **false**. |
| sparse | | Boolean | 对文档中不存在的字段数据不启用索引；这个参数需要特别注意，如果设置为true的话，在索引字段中不会查询出不包含对应字段的文档.。默认值为 **false**. |
| expireAfterSeconds | | integer | 指定一个以秒为单位的数值，完成 TTL设定，设定集合的生存时间。 |
| v | | index version | 索引的版本号。默认的索引版本取决于mongod创建索引时运行的版本。 |
| weights | | document | 索引权重值，数值在 1 到 99,999 之间，表示该索引相对于其他索引字段的得分权重。 |
| default\_language | | string | 对于文本索引，该参数决定了停用词及词干和词器的规则的列表。 默认为英语 |
| language\_override | | string | 对于文本索引，该参数指定了包含在文档中的字段名，语言覆盖默认的language，默认值为 language. |

EX:在后台创建索引：

db.values.ensureIndex({open: 1, close: 1}, {background: true})

通过在创建索引时加background:true 的选项，让创建工作在后台执行

# MongoDB 聚合

MongoDB中聚合(aggregate)主要用于处理数据(诸如统计平均值,求和等)，并返回计算后的数据结果。有点类似sql语句中的 count(\*)

## aggregate() 方法

MongoDB中聚合的方法使用aggregate()。

aggregate() 方法的基本语法格式如下所示：

>db.COLLECTION\_NAME.aggregate(AGGREGATE\_OPERATION)

下表展示了一些聚合的表达式:

|  |  |  |  |
| --- | --- | --- | --- |
| **表达式** | **描述** | | **实例** |
| $sum | | 计算总和。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : "$likes"}}}]) |
| $avg | | 计算平均值 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$avg : "$likes"}}}]) |
| $min | | 获取集合中所有文档对应值得最小值。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$min : "$likes"}}}]) |
| $max | | 获取集合中所有文档对应值得最大值。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$max : "$likes"}}}]) |
| $push | | 在结果文档中插入值到一个数组中。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", url : {$push: "$url"}}}]) |
| $addToSet | | 在结果文档中插入值到一个数组中，但不创建副本。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", url : {$addToSet : "$url"}}}]) |
| $first | | 根据资源文档的排序获取第一个文档数据。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", first\_url : {$first : "$url"}}}]) |
| $last | | 根据资源文档的排序获取最后一个文档数据 | db.mycol.aggregate([{$group : {\_id : "$by\_user", last\_url : {$last : "$url"}}}]) |

## 管道的概念

管道在Unix和Linux中一般用于将当前命令的输出结果作为下一个命令的参数。

MongoDB的聚合管道将MongoDB文档在一个管道处理完毕后将结果传递给下一个管道处理。管道操作是可以重复的。

表达式：处理输入文档并输出。表达式是无状态的，只能用于计算当前聚合管道的文档，不能处理其它的文档。

这里我们介绍一下聚合框架中常用的几个操作：

* $project：修改输入文档的结构。可以用来重命名、增加或删除域，也可以用于创建计算结果以及嵌套文档。
* $match：用于过滤数据，只输出符合条件的文档。$match使用MongoDB的标准查询操作。
* $limit：用来限制MongoDB聚合管道返回的文档数。
* $skip：在聚合管道中跳过指定数量的文档，并返回余下的文档。
* $unwind：将文档中的某一个数组类型字段拆分成多条，每条包含数组中的一个值。
* $group：将集合中的文档分组，可用于统计结果。
* $sort：将输入文档排序后输出。
* $geoNear：输出接近某一地理位置的有序文档。

# MongoDB 复制（副本集）

MongoDB复制是将数据同步在多个服务器的过程。

复制提供了数据的冗余备份，并在多个服务器上存储数据副本，提高了数据的可用性， 并可以保证数据的安全性。

复制还允许您从硬件故障和服务中断中恢复数据。

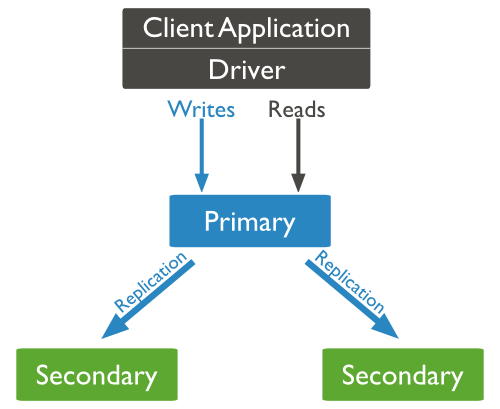
## MongoDB复制原理

mongodb的复制至少需要两个节点。其中一个是主节点，负责处理客户端请求，其余的都是从节点，负责复制主节点上的数据。

mongodb各个节点常见的搭配方式为：一主一从、一主多从。

主节点记录在其上的所有操作oplog，从节点定期轮询主节点获取这些操作，然后对自己的数据副本执行这些操作，从而保证从节点的数据与主节点一致。

MongoDB复制结构图如下所示：



以上结构图中，客户端总主节点读取数据，在客户端写入数据到主节点时， 主节点与从节点进行数据交互保障数据的一致性。

### 副本集特征：

* N 个节点的集群
* 任何节点可作为主节点
* 所有写入操作都在主节点上
* 自动故障转移
* 自动恢复

## MongoDB副本集设置

在本教程中我们使用同一个MongoDB来做MongoDB主从的实验， 操作步骤如下：

1、关闭正在运行的MongoDB服务器。

现在我们通过指定 --replSet 选项来启动mongoDB。--replSet 基本语法格式如下：

mongod --port "PORT" --dbpath "YOUR\_DB\_DATA\_PATH" --replSet "REPLICA\_SET\_INSTANCE\_NAME"

**实例：**

mongod --port 27017 --dbpath "D:\set up\mongodb\data" --replSet rs0

以上实例会启动一个名为rs0的MongoDB实例，其端口号为27017。

启动后打开命令提示框并连接上mongoDB服务。

在Mongo客户端使用命令rs.initiate()来启动一个新的副本集。

我们可以使用rs.conf()来查看副本集的配置

查看副本集姿态使用 rs.status() 命令

## 副本集添加成员

添加副本集的成员，我们需要使用多条服务器来启动mongo服务。进入Mongo客户端，并使用rs.add()方法来添加副本集的成员。

rs.add() 命令基本语法格式如下：

>rs.add(HOST\_NAME:PORT)

**实例：**

假设你已经启动了一个名为mongod1.net，端口号为27017的Mongo服务。 在客户端命令窗口使用rs.add() 命令将其添加到副本集中，命令如下所示：

>rs.add("mongod1.net:27017")

>

MongoDB中你只能通过主节点将Mongo服务添加到副本集中， 判断当前运行的Mongo服务是否为主节点可以使用命令db.isMaster() 。

MongoDB的副本集与我们常见的主从有所不同，主从在主机宕机后所有服务将停止，而副本集在主机宕机后，副本会接管主节点成为主节点，不会出现宕机的情况。

# MongoDB 分片

## 分片

在Mongodb里面存在另一种集群，就是分片技术,可以满足MongoDB数据量大量增长的需求。

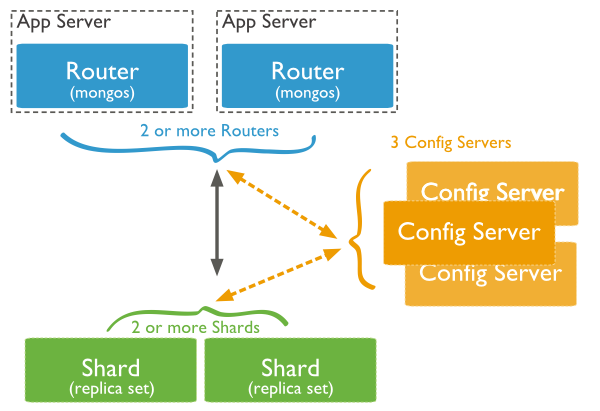
当MongoDB存储海量的数据时，一台机器可能不足以存储数据也足以提供可接受的读写吞吐量。这时，我们就可以通过在多台机器上分割数据，使得数据库系统能存储和处理更多的数据。

## 为什么使用分片

* 复制所有的写入操作到主节点
* 延迟的敏感数据会在主节点查询
* 单个副本集限制在12个节点
* 当请求量巨大时会出现内存不足。
* 本地磁盘不足
* 垂直扩展价格昂贵

## MongoDB分片

下图展示了在MongoDB中使用分片集群结构分布：



上图中主要有如下所述三个主要组件：

* **Shard:**

用于存储实际的数据块，实际生产环境中一个shard server角色可由几台机器组个一个relica set承担，防止主机单点故障

* **Config Server:**

mongod实例，存储了整个 ClusterMetadata，其中包括 chunk信息。

* **Query Routers:**

前端路由，客户端由此接入，且让整个集群看上去像单一数据库，前端应用可以透明使用。

## 分片实例

分片结构端口分布如下：

Shard Server 1：27020

Shard Server 2：27021

Shard Server 3：27022

Shard Server 4：27023

Config Server ：27100

Route Process：40000

步骤一：启动Shard Server

[root@100 /]# mkdir -p /www/mongoDB/shard/s0

[root@100 /]# mkdir -p /www/mongoDB/shard/s1

[root@100 /]# mkdir -p /www/mongoDB/shard/s2

[root@100 /]# mkdir -p /www/mongoDB/shard/s3

[root@100 /]# mkdir -p /www/mongoDB/shard/log

[root@100 /]# /usr/local/mongoDB/bin/mongod --port 27020 --dbpath=/www/mongoDB/shard/s0 --logpath=/www/mongoDB/shard/log/s0.log --logappend --fork

....

[root@100 /]# /usr/local/mongoDB/bin/mongod --port 27023 --dbpath=/www/mongoDB/shard/s3 --logpath=/www/mongoDB/shard/log/s3.log --logappend --fork

步骤二： 启动Config Server

[root@100 /]# mkdir -p /www/mongoDB/shard/config

[root@100 /]# /usr/local/mongoDB/bin/mongod --port 27100 --dbpath=/www/mongoDB/shard/config --logpath=/www/mongoDB/shard/log/config.log --logappend --fork

**注意：**这里我们完全可以像启动普通mongodb服务一样启动，不需要添加—shardsvr和configsvr参数。因为这两个参数的作用就是改变启动端口的，所以我们自行指定了端口就可以。

步骤三： 启动Route Process

/usr/local/mongoDB/bin/mongos --port 40000 --configdb localhost:27100 --fork --logpath=/www/mongoDB/shard/log/route.log --chunkSize 500

mongos启动参数中，chunkSize这一项是用来指定chunk的大小的，单位是MB，默认大小为200MB.

步骤四： 配置Sharding

接下来，我们使用MongoDB Shell登录到mongos，添加Shard节点

[root@100 shard]# /usr/local/mongoDB/bin/mongo admin --port 40000

MongoDB shell version: 2.0.7

connecting to: 127.0.0.1:40000/admin

mongos> db.runCommand({ addshard:"localhost:27020" })

{ "shardAdded" : "shard0000", "ok" : 1 }

......

mongos> db.runCommand({ addshard:"localhost:27029" })

{ "shardAdded" : "shard0009", "ok" : 1 }

mongos> db.runCommand({ enablesharding:"test" }) #设置分片存储的数据库

{ "ok" : 1 }

mongos> db.runCommand({ shardcollection: "test.log", key: { id:1,time:1}})

{ "collectionsharded" : "test.log", "ok" : 1 }

步骤五： 程序代码内无需太大更改，直接按照连接普通的mongo数据库那样，将数据库连接接入接口40000

# MongoDB 备份(mongodump)与恢复(mongorestore)

## MongoDB数据备份

在Mongodb中我们使用mongodump命令来备份MongoDB数据。该命令可以导出所有数据到指定目录中。

mongodump命令可以通过参数指定导出的数据量级转存的服务器。

mongodump命令脚本语法如下：

>mongodump -h dbhost -d dbname -o dbdirectory

* **-h：**

MongDB所在服务器地址，例如：127.0.0.1，当然也可以指定端口号：127.0.0.1:27017

* **-d：**

需要备份的数据库实例，例如：test

* **-o：**

备份的数据存放位置，例如：c:\data\dump，当然该目录需要提前建立，在备份完成后，系统自动在dump目录下建立一个test目录，这个目录里面存放该数据库实例的备份数据。

### 实例

在本地使用 27017 启动你的mongod服务。打开命令提示符窗口，进入MongoDB安装目录的bin目录输入命令mongodump:

>mongodump

执行以上命令后，客户端会连接到ip为 127.0.0.1 端口号为 27017 的MongoDB服务上，并备份所有数据到 bin/dump/ 目录中。

mongodump 命令可选参数列表如下所示：

|  |  |  |
| --- | --- | --- |
| **语法** | **描述** | **实例** |
| mongodump --host HOST\_NAME --port PORT\_NUMBER | 该命令将备份所有MongoDB数据 | mongodump --host w3cschool.cc --port 27017 |
| mongodump --dbpath DB\_PATH --out BACKUP\_DIRECTORY |  | mongodump --dbpath /data/db/ --out /data/backup/ |
| mongodump --collection COLLECTION --db DB\_NAME | 该命令将备份指定数据库的集合。 | mongodump --collection mycol --db test |

## MongoDB数据恢复

mongodb使用 mongorerstore 命令来恢复备份的数据。

mongorestore命令脚本语法如下：

>mongorestore -h dbhost -d dbname --directoryperdb dbdirectory

* **-h：**

MongoDB所在服务器地址

* **-d：**

需要恢复的数据库实例，例如：test，当然这个名称也可以和备份时候的不一样，比如test2

* **--directoryperdb：**

备份数据所在位置，例如：c:\data\dump\test，这里为什么要多加一个test，而不是备份时候的dump，读者自己查看提示吧！

* **--drop：**

恢复的时候，先删除当前数据，然后恢复备份的数据。就是说，恢复后，备份后添加修改的数据都会被删除，慎用哦！

**命令：**

>mongorestore

# MongoDB 监控

在你已经安装部署并允许MongoDB服务后，你必须要了解MongoDB的运行情况，并查看MongoDB的性能。这样在大流量得情况下可以很好的应对并保证MongoDB正常运作。

MongoDB中提供了mongostat 和 mongotop 两个命令来监控MongoDB的运行情况。

## mongostat 命令

mongostat是mongodb自带的状态检测工具，在命令行下使用。它会间隔固定时间获取mongodb的当前运行状态，并输出。如果你发现数据库突然变慢或者有其他问题的话，你第一手的操作就考虑采用mongostat来查看mongo的状态。

启动你的Mongod服务，进入到你安装的MongoDB目录下的bin目录， 然后输入mongostat命令，如下所示：

D:\set up\mongodb\bin>mongostat

## mongotop 命令

mongotop也是mongodb下的一个内置工具，mongotop提供了一个方法，用来跟踪一个MongoDB的实例，查看哪些大量的时间花费在读取和写入数据。 mongotop提供每个集合的水平的统计数据。默认情况下，mongotop返回值的每一秒。

启动你的Mongod服务，进入到你安装的MongoDB目录下的bin目录， 然后输入mongotop命令，如下所示：

D:\set up\mongodb\bin>mongotop

带参数实例

E:\mongodb-win32-x86\_64-2.2.1\bin>mongotop 10

后面的10是*<sleeptime>*参数 ，可以不使用，等待的时间长度，以秒为单位，mongotop等待调用之间。通过的默认mongotop返回数据的每一秒。

E:\mongodb-win32-x86\_64-2.2.1\bin>mongotop --locks

报告每个数据库的锁的使用中，使用mongotop - 锁

输出结果字段说明：

* **ns：**

包含数据库命名空间，后者结合了数据库名称和集合。

* **db：**

包含数据库的名称。名为 . 的数据库针对全局锁定，而非特定数据库。

* **total：**

mongod花费的时间工作在这个命名空间提供总额。

* **read：**

提供了大量的时间，这mongod花费在执行读操作，在此命名空间。

* **write：**

提供这个命名空间进行写操作，这mongod花了大量的时间。