# 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()