MongoDB

可视化工具地址：https://robomongo.org/download

## MongoDB 概念解析

|  |  |  |
| --- | --- | --- |
| **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 |



## 数据库

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

"show dbs" 命令可以显示所有数据的列表。



"db" 命令可以显示当前数据库对象或集合。

"use"命令，可以连接到一个指定的数据库。即切换数据库.

数据库名命名规则：

* 不能是空字符串（"")。
* 不得含有' '（空格)、.、$、/、\和\0 (空字符)。
* 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

文档是一组键值(key-value)对，对应mysql的行。文档不需要设置相同的字段，并且相同字段不需要相同的数据类型，即每一条数据都是独立的，可以有不同的结构。

注意：

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。

## 集合

普通collection：当第一个文档插入时，集合就会被创建。（可以不事先创建）

capped collections（固定大小的collection）：

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。

* 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。

创建集合：

db.createCollection(name, options)

检查创建的集合：

show collections

options选项：

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **描述** |
| capped | Boolean | （可选）如果为true，它启用上限集合。上限集合是一个固定大小的集合，当它达到其最大尺寸会自动覆盖最老的条目。 如果指定true，则还需要指定参数的大小。 |
| autoIndexID | Boolean | （可选）如果为true，自动创建索引\_id字段。默认的值是 false. |
| size | number | （可选）指定的上限集合字节的最大尺寸。如果capped 是true，那么还需要指定这个字段。 |
| max | number | （可选）指定上限集合允许的最大文件数。 |

删除集合：

db.collection.drop()

## 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。 |

## 创建数据库

use 数据库名

若数据库不存在，则创建数据库

## 删除数据库

db.dropDatabase()

## 删除集合

db.collection.drop()

## 插入文档

db.COLLECTION\_NAME.insert(document)

若集合不在当前数据库中，会自动创建该集合并插入文档。

查看文档

db.COLLECTION\_NAME.find()

db.COLLECTION\_NAME.find().pretty()//数据显示结构化，类似于php的pre

db.COLLECTION\_NAME.save(document)

若不指定\_id,则为添加操作

若指定\_id,为更新操作,save的内容会将原有内容全部替换

若指定的\_id 不存在，也会添加

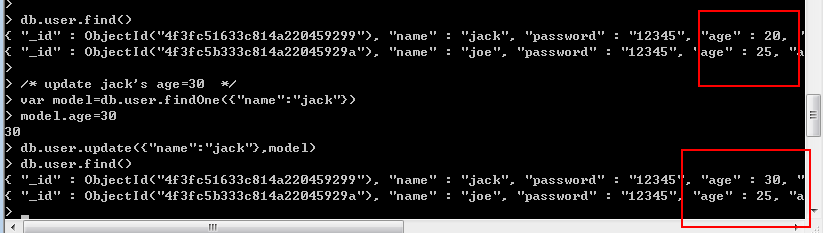
## 更新文档

update语法：将文档中的值更新

|  |
| --- |
| 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。 |

整体更新：

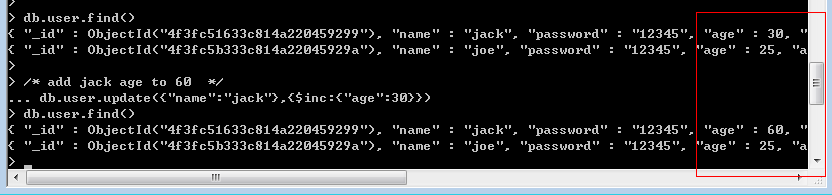
第二个参数会把原有参数覆盖掉



局部更新：（只更新$inc或$set后的字段）

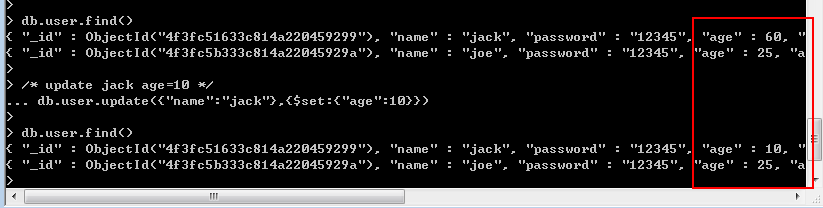
1）$inc修改器

在他原有值的基础上增加或减少



2）$set修改器

直接为字段设置新值



3）$unset修改器

删除某个字段

db.game.update({"\_id":123}, {"$unset": {"author":1}})

4）$push修改器

数组追加

db.game.update({"\_id": 123}, { "$push": {"score": 123}})

亦可追加数组：

db.game.update({"\_id": 123}, {"$push": {"score": [12,123]}})

5）$addToSet tags是数组

db.test.update({"\_id":123 },{"$addToSet":{"tags":222222}})

只有值不在tags里才会添加。类似于集合Set。此时tags应该是个数组。

6）$pop 每次删除数组中的一个元素

db.game.update({"\_id": 123}, {"$pop": {"score": 1}}) // 删除最后一个元素

db.game.update({"\_id": 123}, {"$pop": {"score": -1}}) // 删除第一个元素

7）$pull 删除数组中的特定元素

//00000是tags数组中的一个值

db.test.update({"\_id":123},{"$pull":{"tags":"00000"}})

8）$pullAll 一次删除数组中的多个值

db.test.update({"\_id":123}, {"$pullAll":{"tags":["111", "222"]}})

9）更新嵌套数组的值

{

address: [{place: "nanji", tel: 123}, {place: "dongbei", tel: 321}]

}

db.game.update({"\_id": 123}, {"$set": {"address.0.tel": 213}})

若不知道更新数组是哪一项，用$操作符，（$代表自身的索引，即按查询条件找出的数组里的项本身，且只会找到第一个匹配项）

db.game.update({"address.place": "nanji"}, {"$set": {"address.$.tel": 123}})

save语法：

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。

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

|  |
| --- |
| >db.mycol.save(  {  "\_id" : ObjectId(5983548781331adf45ec7), "title":"Yiibai Yiibai New Topic", "by":"Yiibai Yiibai"  }  )  //替换这个\_id的值 |

## 删除文档

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。默认是全删

writeConcern :（可选）抛出异常的级别。

删除所有（类似truncate）：

db.col.remove({})

## 查询文档

db.collection.find(query, projection)

* **query** ：可选，使用查询操作符指定查询条件
* **projection** ：可选，使用投影操作符指定返回的键。查询时返回文档中所有键值， 只需省略该参数即可（默认省略）。
* db.collection.findOnde(query, projection) //只返回一个

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

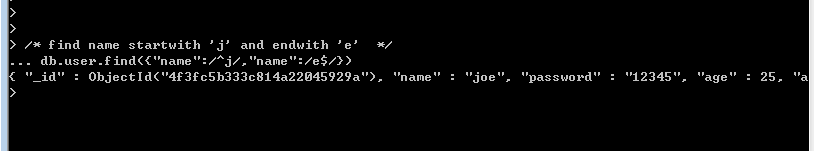
没有查询条件，查询结果中包含title，不包含\_id; 1为显示，0为不显示

查询语句：MongoDB 与 RDBMS Where 语句比较

| **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 |

还有"$in"，"$nin"；{<key>:{$in:[‘a’,’b’]}}

**正则表达式匹配：**



**AND条件**

>db.col.find({key1:value1 **,** key2:value2}).pretty()//多条件用，分割即可

**OR 条件**

>db.col.find(

{

$or: [

{key1: value1}, {key2:value2}

]

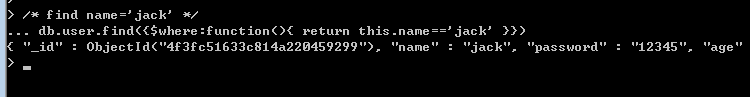
}

).pretty()

**AND 和 OR** 联合使用

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

**$where:**

****

**选择显示字段：**

db.COLLECTION\_NAME.find({},{KEY:1}) 第二个中括号指明字段显示详情，1：显示，0：不显示，指明的字段会显示，没指明的都不显示，但是，\_id永远会显示。

db.COLLECTION\_NAME.find().count() //查询数据总条数

数组查询：

* 1. 数组普通查询

假如type是["mongodb", "javascript"]：

// mongodb在数组type中即可

db.article.find({"type": "mongodb"})

//type数组中只要有一个满足大于4，即可匹配

db.article.find({"type": { $gt : 4}})

//数组type中，至少有一个元素满足后面的条件,同上面类似

也可以用在返回内容的筛选上；

db.article.find({"type": { "$elemMatch": {$eq : “mongodb”}}})

db.article.find({"kown": { "$elemMatch": {a: 1, b: {"$gt": 2}}}})

* 1. 多个元素的数组查询

db.article.find({"type": {"$all": ["mongodb", "javascript"]}})

必须同时包含"mongodb", "javascript"的数组type，才能匹配

区别与：db.article.find({"type":["mongodb", "javascript"]})，它代表完全匹配，而且顺序不能变。

* 1. 限制数组长度查询

db.article.find({"type": {"$size": 2}})

* 1. 返回特定数量

db.test.find(

{"\_id":123)},

{"tags":{"$slice" : num}}

//截取tags数组的num个元素，若为负数，则从后面开始截取

)

db.test.find(

{"\_id":123)},

{"tags":{"$slice" : [start,num]}}

//$slice后面是个数组，截取tags数组的num个元素，从start开始截取

)

* 1. 取模

//要匹配read%5 == 1

db.article.find({"read": {$mod: [5, 1]}})

* 1. 是否存在

//字段love是否存在

db.article.find({"love": {"$exists": true}})

* 1. 正则表达式

db.article.find({"title": /mongodb/i}) // i是忽略大小写

* 1. 内嵌文档（点语法）

//{address: { name: "nanji" }}

db.article.find({"address.name": "nanji"})

数组也可以使用点语法：

//{comments: [{title: "mongodb"}, {title: "javascript"}]}

db.article.find({"comments.title": "mongodb"})

* 1. 取反：

db.article.find({"author": {"$not": /mongodb/i}})

## 查询文档返回字段的部分内容

## 条件操作符

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

db.col.find({‘likes’ : {$lte : 150}})

用< >联合查询

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

## $type数据类型

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 | |  | |

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

## Limit与Skip方法

db.col.find().skip(1).limit(1);

skip参数为从第几条开始，默认从0开始，limit()参数为取多少条

## sort()方法

db.COLLECTION\_NAME.find().sort({KEY:1}) // 1 为升序排列，而-1是用于降序排列。

## 索引

查看索引：

db.collectionName.getIndexes()

创建单列索引：

db.collectionName.ensureIndex({field:1/-1})//1：升序;-1:降序

创建多列索引：

db.collectionName.ensureIndex({field1:1/-1,field2:1/-1})

过期索引：

db.collectionName.ensureIndex({field:1/-1},{expireAfterSeconds:10})//第二参指定过期时间

类型

**全文索引**

(mongodb中每个集合只能创建一个全文索引)

db.articles.ensureIndex({key:’text’});v是固定的

db.articles.ensureIndex({“$\*\*”:’text’});对集合中所有字段进行全文索引

db.article.find({$text:{$search:’搜索关键字 多个关键字用空格分开即可’}}) // ‘或’查询

db.article.find({$text:{$search:’aa bb cc’}}) 只要包含aa bb cc就搜索出来 ，‘或’的关系

db.article.find({$text:{$search:’aa bb -cc’}}) 搜索包含aa 或bb但不包含cc的项

若要是用‘且’的关系

db.article.find({$text:{$search:“\“aa\” \“bb\” \“cc\””}}) 用引号将每个 搜索词引起来，\为转义

相似度:

db.article.find({$text:{$search:’aa bb cc’}},{score:{$meta:”textScore”}}) 在查询结果里追加相似度字段。配合sort排序可以返回按相似度排序的结果:

db.article.find({$text:{$search:’aa bb cc’}},{score:{$meta:”textScore”}}).sort({score:{$meta:”textScore”}})

**ensureIndex({field:1},{ })** 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**.  **当然，若你搜{filed: {$exists: false}} 此时也不会用到这个稀疏索引，而是会全表搜** |
| 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. |
| partialFilterExpression | object | 部分索引: 为满足条件的数据建立索引  db.restaurants.createIndex(  { cuisine: 1, name: 1 },  { partialFilterExpression: { rating: { $gt: 5 } } }  )  db.restaurants.find( { cuisine: "Italian", rating: { $gte: 8 } } ) 会使用索引，因为>=8是>5的子集  db.restaurants.find( { cuisine: "Italian", rating: { $lt: 8 } } ) 不会使用索引 |

A、删除单个索引：

db.collectionName.dropIndex({filed:1/-1})

B、删除所有索引：\_id列的索引不会删除。

db.collectionName.dropIndexes()

索引管理：

* 1. 查看所有索引

db.system.indexes.find()

* 1. 分析查询性能及索引

db.collectionName.find(查询表达式).explain()



"cursor" :"BasicCursor" ——>表示索引没有发挥作用

"nscanned":1 ——>表示查询了多少个文档。

"n"，:1 ——>表示返回的文档数量。

"millis":0 ——>表示整个查询的耗时。

"nscannedObjects" : 11, ——>理论上需要扫描多少行

## MongoDB聚合

聚合操作将多个文档中的值组合在一起，并可对分组数据执行各种操作，以返回单个结果。 在SQL中的 count(\*)与group by组合相当于mongodb 中的聚合功能。

|  |
| --- |
| db.article.insert([  {  \_id: 100,  title: 'MongoDB Overview',  description: 'MongoDB is no sql database',  by\_user: 'Maxsu',  url: 'http://www.yiibai.com',  tags: ['mongodb', 'database', 'NoSQL'],  likes: 100  },  {  \_id: 101,  title: 'NoSQL Overview',  description: 'No sql database is very fast',  by\_user: 'Maxsu',  url: 'http://www.yiibai.com',  tags: ['mongodb', 'database', 'NoSQL'],  likes: 10  },  {  \_id: 102,  title: 'Neo4j Overview',  description: 'Neo4j is no sql database',  by\_user: 'Kuber',  url: 'http://www.neo4j.com',  tags: ['neo4j', 'database', 'NoSQL'],  likes: 750  },  {  \_id: 103,  title: 'MySQL Overview',  description: 'MySQL is sql database',  by\_user: 'Curry',  url: 'http://www.yiibai.com/mysql/',  tags: ['MySQL', 'database', 'SQL'],  likes: 350  }])  db.article.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : 1}}}])  结果：  { "\_id" : "Curry", "num\_tutorial" : 1 }  { "\_id" : "Kuber", "num\_tutorial" : 1 }  { "\_id" : "Maxsu", "num\_tutorial" : 2 }  对于上述用例的Sql等效查询是：  select by\_user, count(\*) as num\_tutorial from `article` group by by\_user; |

| **表达式** | **描述** | **示例** |
| --- | --- | --- |
| $sum | 从集合中的所有文档中求出定义的值。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : "$likes"}}}]) |
| $avg | 计算集合中所有文档的所有给定值的平均值。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$avg : "$likes"}}}]) |
| $min | 从集合中的所有文档获取相应值的最小值。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$min : "$likes"}}}]) |
| $max | 从集合中的所有文档获取相应值的最大值。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$max : "$likes"}}}]) |
| $push | 将值插入到生成的文档中的数组中。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", url : {$push: "$url"}}}]) |
| $addToSet | 将值插入生成的文档中的数组，但不会创建重复项。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", url : {$addToSet : "$url"}}}]) |
| $first | 根据分组从源文档获取第一个文档。 通常情况下，这只适用于以前应用的“$sort”阶段。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", first\_url : {$first : "$url"}}}]) |
| $last | 根据分组从源文档获取最后一个文档。通常情况下，这只适用于以前应用的“$sort”阶段。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", last\_url : {$last : "$url"}}}]) |

## 关联关系

|  |
| --- |
| {  "\_id":ObjectId("52ffc33321332111sdfaf"),  "contact": "13800138000",  "dob": "1991-11-11",  "name": "Maxsu",  "address\_ids": [  ObjectId("123123"),  ObjectId("123412")  ]  }  如上所示，用户文档包含对应地址的ObjectId的数组字段address\_ids。 使用这些ObjectIds，我们可以从那里查询地址文件并获取地址详细信息。 使用这种方法，需要两个查询：首先从用户文档获取address\_ids字段，然后从地址集中获取这些地址。  >var result = db.users.findOne({"name":"Maxsu"},{"address\_ids":1})  >var addresses = db.address.find({"\_id":{"$in":result["address\_ids"]}}) |

## MongoDB DBRefs

用于文档包含来自不同集合的引用；

DBRefs中有三个字段 -

* $ref - 此字段指定引用文档的集合
* $id - 此字段指定引用文档的\_id字段
* $db - 这是一个可选字段，并包含引用文档所在的数据库的名称

|  |
| --- |
| 假设一个具有DBRef字段address的示例用户文档，如代码片段所示 -  {  "\_id":ObjectId("348362491fjaskdlf2314"),  "address": {  "$ref": "address\_home",  "$id": ObjectId("sfaafdf4137832149fssa"),  "$db": "yiibai"},  "contact": "13800138000",  "dob": "1991-12-12",  "name": "Maxsu"  }  address指定引用的地址文件位于yiibai数据库中的address\_home集合中，其ID为sfaafdf4137832149fssa。  >var user = db.users.findOne({"name":"Maxsu"})  >var dbRef = user.address  >db[dbRef.$ref].findOne({"\_id":(dbRef.$id)})  返回：{  "\_id" : ObjectId("sfaafdf4137832149fssa"),  "building" : "Hainan Apt No.2100",  "pincode" : 571100,  "city" : "Haikou",  "province" : "Hainan"  } |

## 分析查询

explain()

db.users.find({gender:"M"},{user\_name:1,\_id:0}).explain(‘executionStats’)

* indexOnly的true值表示此查询已使用索引。
* cursor字段指定使用的游标的类型。BTreeCursor类型表示使用了索引，并且还给出了使用的索引的名称。 BasicCursor表示完全扫描，而不使用任何索引的情况。
* n表示返回的文档数。
* nscannedObjects表示扫描的文档总数。
* nscanned表示扫描的文档或索引条目的总数。
* totalKeysExamined索引扫描条数
* totalDocsExamined文档扫描条数

理想情况，nReturned=totalKeysExamined & totalDocsExamined=0

不通过文档查询，全部使用索引

sort的顺序跟创建索引的顺序必须一致，否则不会生效！！！

索引是{filed1: 1, field2:1}， sort({fileld1: 1, filed2: 1}) 可命中索引，sort({filed2:1,filed1:1})索引无效

hint()

$hint操作符强制查询优化器使用指定的索引来运行查询。当要测试具有不同索引的查询的性能时，这就特别有用了。 例如，以下查询指定要用于此查询的gender和user\_name字段的索引

> db.users.find({gender:"M"},{user\_name:1,\_id:0}).hint({gender:1,user\_name:1})

要使用$explain来分析上述查询 -

>db.users.find({gender:"M"},{user\_name:1,\_id:0}).hint({gender:1,user\_name:1}).explain()

## 原子操作

MongoDB不支持多文档原子事务。 但是，它可以为单个文档提供了原子操作。

如果文档有一百个字段，则更新语句将要么全部更新，要么全部不更新所选字段的值，因此在原始级别保持原子性。

维持原子性的推荐方法是将所有相关信息保存在一起，并使用嵌入式文档在一个文档中一起更新。

|  |
| --- |
| {  "\_id":1,  "product\_name": "Huawei P9",  "category": "mobiles",  "product\_total": 5,  "product\_available": 3,  "product\_bought\_by": [  {  "customer": "Kobe",  "date": "2017-07-08"  },  {  "customer": "Maxsu",  "date": "2018-07-28"  }  ]  } |

当有新顾客购买时，先检查product\_available是否大于0，若大于0则减少product\_available,并将新顾客的信息插入到product\_bought\_by中。

|  |
| --- |
| >db.products.findAndModify({  query:{\_id:2,product\_available:{$gt:0}},  update:{  $inc:{product\_available:-1},  $push:{product\_bought\_by:{customer:"Curry",date:"2017-08-08"}}  }  }) |

若不适用嵌入式，而将库存和购物者信息放在不同集合中，可能会造成0库存但还能购买的情况。因为这种操作不是原子性的。

## 高级索引

**索引数组字段：**

|  |
| --- |
| {  "address": {  "city": "Haikou",  "province": "Hainan",  "pincode": "123456"  },  "tags": [  "music",  "cricket",  "blogs"  ],  "name": "Maxsu"  } |

要在tags数组上创建索引，请使用以下代码 -

>db.users.ensureIndex({"tags":1})

创建索引后，可以搜索这个集合的标签字段 -

>db.users.find({tags:"cricket"})

**索引子文档字段：**

子文档即json里面又套有json；

在上面的文档中的子文档上创建索引

>db.users.ensureIndex({"address.city":1,"address.state":1,"address.pincode":1})

创建索引后，就可以使用此索引来搜索任何子文档的字段了，如下所示：

>db.users.find({"address.city":"Haikou"})

## PHP操作MongoDB

<http://www.jb51.net/Special/586.htm>

## ObjectId 解析

\_id: ObjectId(' 4e7020cb7cac81af7136236b ')

4e7020cb7cac81af7136236b 这个24位字符串，是一组十六进制字符串，每个字节2位的十六进制，所以一共是12个字节；



一共有四部分：

1. time

前4个字节 " 4e7020cb ",转成十进制，即：" 1315971275 "，是个时间戳

2）machine

接下来是" 7cac81 ",代表所在主机的唯一标识符，一般是机器主机名的散列值，确保了不同主机生成不同的机器hash值，确保在分布式中不造成冲突，所以在同一个机器上生成的ObjectId中间的字符串都是一样的

3）pid

进程Id，确保同一台机器不同的mongodb进程产生的ObjectId不冲突，接下来的两位" af71 "，就是pid

4）inc

自增计数器，前面的九个字节是保证了一秒内不同机器不同进程生成objectId不冲突，这后面的三个字节“36236b”是一个自动增加的计数器，用来确保在同一秒内产生的objectId也不会发现冲突，允许256的3次方等于16777216条记录的唯一性。