第一章

Mongodb是面向文档的数据库，不是关系型数据库，方便了可扩展性。

不必键所有数据都放到一个模子里面

Mongodb使用mongodb传输协议作为与服务器交互的主要方式。

动态查询优化器

1. 入门

每一个文档都有一个特殊的键\_id，它在文档中所处的集合中是唯一的

Mongodb不但区分类型，也区分大小写：

{“foo”:2}和{“foo”:”2”}和{“Foo”:2}都不同

Mongodb不能有重复的键：重复的键会取第一个，之后的重复项会自动略去

Mongodb中的为集合collection,表示一组文档，类似于关系型数据库中的表

Mongodb中的集合是无模式的，这意味着集合里面的文档可以是任何格式的：

{“a”:1},{“a”:2,”b”:2}

组织集合的一种惯例是使用点操作符，如blog.posts和blog.autors对应两个集合，这里的blog不是不需的，只是为了表示这两个集合与blog有关

要记住，数据库名最终会变成文件系统里的文件，因此数据库的名字有很多的限制。

集合的命名空间就是：数据库名.集合名，命名空间的长度不能超过121个字节

端口号是：27017

Mongodb自带javascript的shell，这里可运行任何的javascript程序，还可以充分利用javascript的标准库

> x=200

200

> x/5

40

> Math.sin(Math.PI/2);

1

> new Date("2010/1/1");

ISODate("2009-12-31T16:00:00Z")

> "hello world".replace("world","mongodb")

hello mongodb

而且还可以定义和调用javascript函数

> function s(n){ if(n<=1) return 1; return n\*s(n-1); }

> s(20)

2432902008176640000

> s(10)

3628800

2.5.3 shell中的基本操作

1. 创建--C

> post={"title":"my blog post.",//使用局部变量post

... "content":"here's my blog post.",

... "date":new Date()}

{

"title" : "my blog post.",

"content" : "here's my blog post.",

"date" : ISODate("2016-10-16T13:36:06.649Z")

}

> db.blog.insert(post)

WriteResult({ "nInserted" : 1 })

> db.blog.find()

{ "\_id" : ObjectId("580382555a808d2420e8fa54"), "title" : "my blog post.", "content" : "here's my blog po

st.", "date" : ISODate("2016-10-16T13:36:06.649Z") }

1. 查找--R

Find:返回所有的文档，findOne只返回一个文档

1. 更新--U

Update

> post

{

"title" : "my blog post.",

"content" : "here's my blog post.",

"date" : ISODate("2016-10-16T13:36:06.649Z")

}

> post.comments=[]

[ ]

> db.blog.update({"title":"my blog post."},post)

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.blog.find()

{ "\_id" : ObjectId("580382555a808d2420e8fa54"), "title" : "my blog post.", "content" : "here's my blog po

st.", "date" : ISODate("2016-10-16T13:36:06.649Z"), "comments" : [ ] }

1. 删除--D

Remove函数

查看帮助;

Db.help()查看数据库级别的命令帮助

Db.test（集合名）.help()查看集合级别的命令帮助

小窍门：在写命令的时候，不加括号则返回该函数的javascript源代码

> db.test.update//不输入括号则输出函数的源代码

function (query, obj, upsert, multi) {

var parsed = this.\_parseUpdate(query, obj, upsert, multi);

var query = parsed.query;

var obj = parsed.obj;

var upsert = parsed.upsert;

var multi = parsed.multi;

var wc = parsed.wc;

var result = undefined;

var startTime =

(typeof(\_verboseShell) === 'undefined' || !\_verboseShell) ? 0 : new Date().getTime();

if (this.getMongo().writeMode() != "legacy") {

var bulk = this.initializeOrderedBulkOp();

var updateOp = bulk.find(query);

if (upsert) {

updateOp = updateOp.upsert();

}

if (multi) {

updateOp.update(obj);

} else {

updateOp.updateOne(obj);

}

try {

result = bulk.execute(wc).toSingleResult();

} catch (ex) {

if (ex instanceof BulkWriteError || ex instanceof WriteCommandError) {

result = ex.toSingleResult();

} else {

// Other exceptions thrown

throw Error(ex);

}

}

} else {

this.\_validateUpdateDoc(obj);

this.getMongo().update(this.\_fullName, query, obj, upsert, multi);

// enforce write concern, if required

if (wc)

result = this.runCommand("getLastError", wc instanceof WriteConcern ? wc.toJSON() : wc

);

}

this.\_printExtraInfo("Updated", startTime);

return result;

}

> db.version

function () {

return this.serverBuildInfo().version;

}

> db.version()

3.2.9

当集合名与mongodb的属性名冲突时，可使用getcollection函数

> db.getCollection("version")

test.version

2.6 数据类型

JSON仅仅包括6种数据类型，为null 布尔 数字 字符串 数组和对象（即内嵌一个json）

Mongodb支持的几种数据类型：

Null:用来表示空值或者不存在的字段{“x”:null}

布尔类型：true 和 false

{“x”:true}

shell中的数字类型都是64为浮点型

对象id，文档的12字节唯一ID

日期:存储的是从标准纪元开始的毫秒数，不存储时区

{“x”:new Date()}

> post={"time":new Date()}

{ "time" : ISODate("2016-10-16T14:00:13.017Z") }

文档中可以包含正则表达式，采用javascript的正则表达式语法

{“x”:/foobar/i}

> reg={"x":/foobar/i}

{ "x" : /foobar/i }

> reg

{ "x" : /foobar/i }

代码:文档中还可以包含javascript代码

> code={"x":function(){/\*...\*/}}

{ "x" : function (){/\*...\*/} }

> code

{ "x" : function (){/\*...\*/} }

构造函数Date()和对象new Date()是不同的，

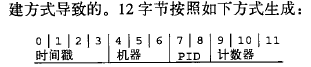
> post={"time":Date()}

{ "time" : "Sun Oct 16 2016 22:08:16 GMT+0800" }

2.6.6 \_id和ObjectId

mongodb中存储的文档必须有一个\_id键，这个键的值可以是任何类型的。默认是个ObjectId对象

生成方式：



1. ：创建、更新及删除文档

3.1 在集合中插入并保存文档

> db.foo.insert({'name':'tony'})

由于消息头的存在，批量插入优于一个一个的插入

3.1.2 插入：原理和作用

驱动程序会将数据转换为bson格式，并检查是否包含\_id键和文档大小是否超过4MB，然后将数据原样插入到数据库，好的是防止注入攻击，不好的是允许插入无效的数据。

可以通过启动--objcheck来告诉服务器在插入之前检查文档的有效性，但会消耗部分性能。

查看文档doc转换为bson的大小，可以在shell中使用命令：Object.bsonsize(doc)

Mongodb在插入的时候并不执行代码，因此几乎没有注入攻击的问题

3.2 删除文档

> db.users.remove({“name”:”tony”})

从数据库中删除整个集合：

> db.foo.drop()

删除整个集合中的所有文档，但保留集合：

> db.foo.remove("")

查看帮助：

Help

查看集合的命令帮助：db.mycoll.help()

3.3 更新文档

更新操作时原子的，从而不必担心更新的冲突问题，先到达的先更新，然后在执行后到达的更新

> db.foo.findOne({"name":"tony"})

利用bson格式，定义更新文档并执行更新操作：

> var joe=db.foo.findOne({"name":"tony"});//定义一个变量joe，是查询的结果，为bson格式

> joe.relations={"friends":"32","enemies":"1"};//增加一个属性relations和值

{ "friends" : "32", "enemies" : "1" }

> joe.username=joe.name;//增加属性并复制为joe.name

tony

> db.foo.update({"name":"tony"},joe)

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.foo.findOne({"name":"tony"})

{

"\_id" : ObjectId("58021ea15a808d2420e8f90b"),

"name" : "tony",

"relations" : {

"friends" : "32",

"enemies" : "1"

},

"username" : "tony"

}

通过bson变量joe来查询相关的数据：

> joe.name

tony

> joe.relations

{ "friends" : "32", "enemies" : "1" }

> joe.id

> joe.username

Tony

更新操作可能在查询出多个文档并执行更新操作时存在错误，但目前好像大部分的程序并不会报此错误

> db.foo.insert({"name":"tony"})//再次插入一个name为tony的文档

WriteResult({ "nInserted" : 1 })

> db.foo.findOne({"name":"tony"})//此命令查询到一个

{

"\_id" : ObjectId("58021ea15a808d2420e8f90b"),

"name" : "tony",

"relations" : {

"friends" : "32",

"enemies" : "1"

},

"username" : "tony"

}

> db.foo.find()//这命令将所有的文档都查询了出来

{ "\_id" : ObjectId("58021ea15a808d2420e8f90b"), "name" : "tony", "relations" : { "friends"

: "32", "enemies" : "1" }, "username" : "tony" }

{ "\_id" : ObjectId("5802213a5a808d2420e8f90c"), "name" : "tony" }

> db.foo.find({"name":"tony"})//查询出所有匹配的文档，这里有两个

{ "\_id" : ObjectId("58021ea15a808d2420e8f90b"), "name" : "tony", "relations" : { "friends"

: "32", "enemies" : "1" }, "username" : "tony" }

{ "\_id" : ObjectId("5802213a5a808d2420e8f90c"), "name" : "tony" }

为了避免更新错误，最好确保更新指定的唯一文档，例如使用\_id这样的键来匹配

3.3.2 使用修改器

由于每次只是对文档的部分数据进行更新，因此使用修改器会更加的高效，而不是每次都更新整个文档。

> db.foo.drop()

true

> show tables

> db.test.insert({"url":"www.baidu.com","pageviews":10})

WriteResult({ "nInserted" : 1 })

> show tables

test

> db.test.findOne()

{

"\_id" : ObjectId("5802258f5a808d2420e8f90e"),

"url" : "www.baidu.com",

"pageviews" : 10

}

> db.test.update({"url":"www.baidu.com"},

... {"$inc":{"pageviews":1}})//使用修改器$inc，增加数值为1

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.test.findOne()

{

"\_id" : ObjectId("5802258f5a808d2420e8f90e"),

"url" : "www.baidu.com",

"pageviews" : 11

}

PHP程序员在写相关的修改器命令时，要将$转义以避免被当做变量值，或者在配置文件中将驱动选项mongo.cmd\_char设置成其他的值即可

使用修改器时，\_id的值不能改变（而在整个文档替换时是可以改变\_id的，因此整个文替换时可以先去掉更新文档里面的\_id项）

修改器;

$set：指定一个键的值，如果键不存在则创建

> db.test.insert({"name":"joe","age":30,"sex":"male","location":"wisconsin"})

WriteResult({ "nInserted" : 1 })

> db.test.findOne({"name":"joe"})

{

"\_id" : ObjectId("5802283f5a808d2420e8f90f"),

"name" : "joe",

"age" : 30,

"sex" : "male",

"location" : "wisconsin"

}

> db.test.update({"name":"joe"},

... {"$set":{"age":31,"favorite book":"war and peace"}}//不存在的则创建

... )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.test.findOne({"name":"joe"})

{

"\_id" : ObjectId("5802283f5a808d2420e8f90f"),

"name" : "joe",

"age" : 31,

"sex" : "male",

"location" : "wisconsin",

"favorite book" : "war and peace"

}

$set还可以修改键的数据类型，如将favorite book键值改为一个数组：

> db.test.update({"name":"joe"},

... {"$set":{"age":30,"favorite book":["book1","book2"]}}//改为数组类型

... )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.test.findOne({"name":"joe"})

{

"\_id" : ObjectId("5802283f5a808d2420e8f90f"),

"name" : "joe",

"age" : 30,

"sex" : "male",

"location" : "wisconsin",

"favorite book" : [

"book1",

"book2"

]

}

$unset可以删除键;

> db.test.update({"name":"joe"},

... {"$unset":{"age":1}}//删除age键

... )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.test.findOne({"name":"joe"})

{

"\_id" : ObjectId("5802283f5a808d2420e8f90f"),

"name" : "joe",

"sex" : "male",

"location" : "wisconsin",

"favorite book" : [

"book1",

"book2"

]

}

修改内嵌文档;

> db.blog.posts.insert({"title":"a blog post","content":"...","author":{"name":"joe","email":"joe@example.com"}})

WriteResult({ "nInserted" : 1 })

> db.blog.posts.findOne({"author.name":"joe"})//使用内嵌值来查询

{

"\_id" : ObjectId("58022b485a808d2420e8f910"),

"title" : "a blog post",

"content" : "...",

"author" : {

"name" : "joe",

"email" : "joe@example.com"

}

}

> db.blog.posts.update({"author.name":"joe"}, {"$set":{"author.name":"joe schmoe"}})//更新内嵌文档

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.blog.posts.findOne()

{

"\_id" : ObjectId("58022b485a808d2420e8f910"),

"title" : "a blog post",

"content" : "...",

"author" : {

"name" : "joe schmoe",

"email" : "joe@example.com"

}

}

增加和减少：

$inc,用来增加已经存在的键的值，如果键不存在则创建，该键必须是数字类型，要修改该数据类型可以使用$set修改器

> db.games.insert({"game":"pinball","user":"joe"})

WriteResult({ "nInserted" : 1 })

> db.games.findOne({"game":"pinball"})

{

"\_id" : ObjectId("58022d6e5a808d2420e8f911"),

"game" : "pinball",

"user" : "joe"

}

> db.games.update({"game":"pinball"},

... {"$inc":{"score":50}}//不存在时创建

... )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.games.findOne()

{

"\_id" : ObjectId("58022d6e5a808d2420e8f911"),

"game" : "pinball",

"user" : "joe",

"score" : 50

}

数组修改器

1. 只能用在值为数组的键上
2. $pus会向已经存在键的数组值的末尾加入一个元素，如果键不存在则此键，并设置值为数组

> db.blog.posts.update({"title":"a blog post"},

... {$push:{"comments":{"name":"joe","email":"joe@example.com","contnet":"nice post."}}}//$push可以不用双引号括起来

... )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.blog.posts.findOne()

{

"\_id" : ObjectId("58022b485a808d2420e8f910"),

"title" : "a blog post",

"content" : "...",

"author" : {

"name" : "joe schmoe",

"email" : "joe@example.com"

},

"comments" : [

{

"name" : "joe",

"email" : "joe@example.com",

"contnet" : "nice post."

}

]

}

再增加条评论：

> db.blog.posts.update({"title":"a blog post"},

... {$push:{"comments":"good post."}}//增加的格式不需要一致

... )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.blog.posts.findOne()

{

"\_id" : ObjectId("58022b485a808d2420e8f910"),

"title" : "a blog post",

"content" : "...",

"author" : {

"name" : "joe schmoe",

"email" : "joe@example.com"

},

"comments" : [

{

"name" : "joe",

"email" : "joe@example.com",

"contnet" : "nice post."

},

"good post."

]

}

> db.blog.posts.update({"comments":{$ne:"good post"}},

... {$push:{"comments":"very good job."}}

... )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.blog.posts.findOne()

{

"\_id" : ObjectId("58022b485a808d2420e8f910"),

"title" : "a blog post",

"content" : "...",

"author" : {

"name" : "joe schmoe",

"email" : "joe@example.com"

},

"comments" : [

{

"name" : "joe",

"email" : "joe@example.com",

"contnet" : "nice post."

},

"good post.",

"very good job."

]

}

可以使用在查询时使用$ne来查询，当数组值中不存在时被检索出来，

db.blog.posts.findOne()

{

"\_id" : ObjectId("58022b485a808d2420e8f910"),

"title" : "a blog post",

"content" : "...",

"author" : {

"name" : "joe schmoe",

"email" : "joe@example.com"

},

"comments" : [

{

"name" : "joe",

"email" : "joe@example.com",

"contnet" : "nice post."

},

"good post."

]

}

> db.blog.posts.update({"comments":{$ne:"good post"}},//good post不在则被检索，更新数据

... {$push:{"comments":"very good job."}}

... )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.blog.posts.findOne()

{

"\_id" : ObjectId("58022b485a808d2420e8f910"),

"title" : "a blog post",

"content" : "...",

"author" : {

"name" : "joe schmoe",

"email" : "joe@example.com"

},

"comments" : [

{

"name" : "joe",

"email" : "joe@example.com",

"contnet" : "nice post."

},

"good post.",

"very good job."

]

}

> db.blog.posts.update({"comments":{$ne:"good post."}}, {$push:{"comments":"very very good jo

b."}} )//good post.存在则不被更新

WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })

> db.blog.posts.findOne()

{

"\_id" : ObjectId("58022b485a808d2420e8f910"),

"title" : "a blog post",

"content" : "...",

"author" : {

"name" : "joe schmoe",

"email" : "joe@example.com"

},

"comments" : [

{

"name" : "joe",

"email" : "joe@example.com",

"contnet" : "nice post."

},

"good post.",

"very good job."

]

}

$addToSet也可以完成同样的事情，而且能够避免重复，即数组存在则不添加：

> db.blog.posts.update({"title":"a blog post"},

... {$addToSet:{"comments":"can be added in."}}//不存在则添加

... )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.blog.posts.findOne()

{

"\_id" : ObjectId("58022b485a808d2420e8f910"),

"title" : "a blog post",

"content" : "...",

"author" : {

"name" : "joe schmoe",

"email" : "joe@example.com"

},

"comments" : [

{

"name" : "joe",

"email" : "joe@example.com",

"contnet" : "nice post."

},

"good post.",

"very good job.",

"can be added in."

]

}

> db.blog.posts.update({"title":"a blog post"}, {$addToSet:{"comments":"good post."}})//存在则并不会添加

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })

> db.blog.posts.findOne()

{

"\_id" : ObjectId("58022b485a808d2420e8f910"),

"title" : "a blog post",

"content" : "...",

"author" : {

"name" : "joe schmoe",

"email" : "joe@example.com"

},

"comments" : [

{

"name" : "joe",

"email" : "joe@example.com",

"contnet" : "nice post."

},

"good post.",

"very good job.",

"can be added in."

]

}

将$addToSet和$each组合起来，可以添加多个不同的值

> db.blog.posts.update({"title":"a blog post"},

... {$addToSet:{"comments":{$each:["a","b","c"]}}}//4each起到遍历数组的作用，因此他后面的数据要是一个数组

... )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.blog.posts.findOne()

{

"\_id" : ObjectId("58022b485a808d2420e8f910"),

"title" : "a blog post",

"content" : "...",

"author" : {

"name" : "joe schmoe",

"email" : "joe@example.com"

},

"comments" : [

{

"name" : "joe",

"email" : "joe@example.com",

"contnet" : "nice post."

},

"good post.",

"very good job.",

"can be added in.",

"a",

"b",

"c"

]

}

数组元素的删除：

$pop可以从数组的任何一段删除一个元素，

> db.blog.posts.update({"title":"a blog post"},

... {$pop:{"comments":1}}//从数组末尾删除一个元素，{$pop:{key:1}}从数组末尾删除一个元素，{$pop:{key:-1}}从数组开头删除一个元素

... )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.blog.posts.findOne()

{

"\_id" : ObjectId("58022b485a808d2420e8f910"),

"title" : "a blog post",

"content" : "...",

"author" : {

"name" : "joe schmoe",

"email" : "joe@example.com"

},

"comments" : [

{

"name" : "joe",

"email" : "joe@example.com",

"contnet" : "nice post."

},

"good post.",

"very good job.",

"can be added in.",

"a",

"b"

]

}

> db.blog.posts.update({"title":"a blog post"}, {$pop:{"comments":-1}} )//从数组开头删除一个元素

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.blog.posts.findOne()

{

"\_id" : ObjectId("58022b485a808d2420e8f910"),

"title" : "a blog post",

"content" : "...",

"author" : {

"name" : "joe schmoe",

"email" : "joe@example.com"

},

"comments" : [

"good post.",

"very good job.",

"can be added in.",

"a",

"b"

]

}

$pull可以删除数组元素的指定元素，$pull会将数组中所有匹配的元素删除掉

> db.blog.posts.update({"title":"a blog post"},

... {$pull:{"comments":"very good job."}}//删除掉comments里面的指定元素

... )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.blog.posts.findOne()

{

"\_id" : ObjectId("58022b485a808d2420e8f910"),

"title" : "a blog post",

"content" : "...",

"author" : {

"name" : "joe schmoe",

"email" : "joe@example.com"

},

"comments" : [

"good post.",

"can be added in.",

"a",

"b"

]

}

> db.blog.posts.update({"title":"a blog post"}, {$pull:{"comments":"very good job haha"}} )//不存在这个元素的话则不进行任何操作

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })

> db.blog.posts.findOne()

{

"\_id" : ObjectId("58022b485a808d2420e8f910"),

"title" : "a blog post",

"content" : "...",

"author" : {

"name" : "joe schmoe",

"email" : "joe@example.com"

},

"comments" : [

"good post.",

"can be added in.",

"a",

"b"

]

}

数组的定位修改器：

有两种方法操作数组中的值：通过位置或者通过定位操作符”$”

数组都是以0 开头的，可以将下标直接作为键来选择元素

> db.blog.posts.insert({"title":"数组下标测试","content":".."})

WriteResult({ "nInserted" : 1 })

> db.blog.posts.findOne({"content":".."})

{

"\_id" : ObjectId("58023ca55a808d2420e8f912"),

"title" : "数组下标测试",

"content" : ".."

}

> db.blog.posts.update({"content":".."},

... {$push:{"comments":{"comment":"good post","author":"john","votes":0}}}

... )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.blog.posts.update({"content":".."},

... {$push:{"comments":{$each:[{"comment":"i thought it was too short","author":"clarie","vot

es":3},{"comment":"free watches","author":"alice","votes":-1}]}}}

... )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.blog.posts.findOne({"content":".."})

{

"\_id" : ObjectId("58023ca55a808d2420e8f912"),

"title" : "数组下标测试",

"content" : "..",

"comments" : [

{

"comment" : "good post",

"author" : "john",

"votes" : 0

},

{

"comment" : "i thought it was too short",

"author" : "clarie",

"votes" : 3

},

{

"comment" : "free watches",

"author" : "alice",

"votes" : -1

}

]

}

> db.blog.posts.update({"content":".."},

... {$inc:{"comments.0.votes":2}}//给第一个元素的vote的值增加2

... )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.blog.posts.findOne({"content":".."})

{

"\_id" : ObjectId("58023ca55a808d2420e8f912"),

"title" : "数组下标测试",

"content" : "..",

"comments" : [

{

"comment" : "good post",

"author" : "john",

"votes" : 2

},

{

"comment" : "i thought it was too short",

"author" : "clarie",

"votes" : 3

},

{

"comment" : "free watches",

"author" : "alice",

"votes" : -1

}

]

}

> db.blog.posts.findOne({"comments.author":"john"})//不指定数组的下标也能查询到数据

{

"\_id" : ObjectId("58023ca55a808d2420e8f912"),

"title" : "数组下标测试",

"content" : "..",

"comments" : [

{

"comment" : "good post",

"author" : "john",

"votes" : 2

},

{

"comment" : "i thought it was too short",

"author" : "clarie",

"votes" : 3

},

{

"comment" : "free watches",

"author" : "alice",

"votes" : -1

}

]

}

使用$定位查询文档已经匹配的元素，需要注意的是$定位符只更新第一个数据，因此有多个匹配的文档时除了第一个其他的并没有更新

> db.blog.posts.update({"comments.author":"john"}, {$set:{"comments.$.author":"jim"}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.blog.posts.findOne({"comments.author":"jim"})

{

"\_id" : ObjectId("58023ca55a808d2420e8f912"),

"title" : "数组下标测试",

"content" : "..",

"comments" : [

{

"comment" : "good post",

"author" : "jim",

"votes" : 2

},

{

"comment" : "i thought it was too short",

"author" : "clarie",

"votes" : 3

},

{

"comment" : "free watches",

"author" : "alice",

"votes" : -1

}

]

}

仅仅需要修改值像$inc的修改速度非常快，但是如果改变了文档的大小就相对较慢。

通常mongodb操作数组时会相对较慢

Upsert,先查找，找到则更新，没有找到则创建,update的第三个参数为true则表示这是个upsert

> db.analytics.update({"url":"/blog"},{$inc:{"vistis":1}},true)

WriteResult({

"nMatched" : 0,

"nUpserted" : 1,

"nModified" : 0,

"\_id" : ObjectId("580247b2b58a9eca0cf5c9ab")

})

> db.analytics.findOne({"url":"/blog"})

{

"\_id" : ObjectId("580247b2b58a9eca0cf5c9ab"),

"url" : "/blog",

"vistis" : 1

}

> db.math.update({"count":25},{$inc:{"count":2}},true)//不存在时将以查询条件创建，并在此基础上增加2

WriteResult({

"nMatched" : 0,

"nUpserted" : 1,

"nModified" : 0,

"\_id" : ObjectId("58024856b58a9eca0cf5c9ac")

})

> db.math.findOne()

{ "\_id" : ObjectId("58024856b58a9eca0cf5c9ac"), "count" : 27 }

Save函数，可以在文档不存在时插入，存在时更新。他只有一个参数：文档。当这个文档存在\_id键时会调用upsert方法，否则会调用插入，可见关键是\_id键的有无。

目前默认情况下，更新只能对符合匹配条件的第一个文档执行操作，要是有多个文档符合条件，其余的文档就没有变化，要使所有的文档都得到更新，可以设置update的第四个参数为true.(今后可能更改，以当时的版本为准)

> db.test.find()

{ "\_id" : ObjectId("5802258f5a808d2420e8f90e"), "url" : "www.baidu.com", "pageviews" : 11 }

{ "\_id" : ObjectId("5802283f5a808d2420e8f90f"), "name" : "joe", "sex" : "male", "location" :

"wisconsin", "favorite book" : [ "book1", "book2" ] }

{ "\_id" : ObjectId("58024b8e5a808d2420e8f913"), "url" : "www.baidu.com" }

> db.test.update({"url":"www.baidu.com"},{$inc:{"pageviews":2}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.test.find({"url":"www.baidu.com"})

{ "\_id" : ObjectId("5802258f5a808d2420e8f90e"), "url" : "www.baidu.com", "pageviews" : 13 }

{ "\_id" : ObjectId("58024b8e5a808d2420e8f913"), "url" : "www.baidu.com" }

> db.test.update({"url":"www.baidu.com"},{$inc:{"pageviews":2}},false,true)

查询最近一次操作的状态：

> db.runCommand({getLastError:1})

{

"connectionId" : 10,

"n" : 0,

"syncMillis" : 0,

"writtenTo" : null,

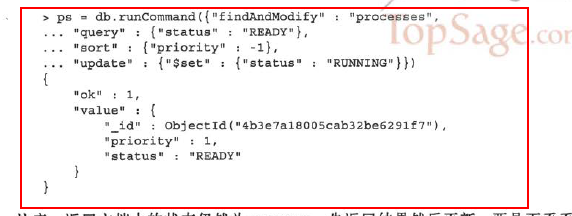
"err" : null,

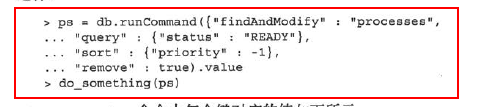
"ok" : 1

}

findAndModify：

1、查找并更新在一个操作内完成。

2、查找并删除：



findAndModify命令中每个键对应的值如下：

1. findAndModify

字符串，集合名

1. Query

查询文档，用来检索文档的条件

1. Sort

排序结果的条件

1. Update

修改器文档，对所找到的文档执行的更新

1. Remove

布尔类型，表示是否删除文档

1. New

布尔类型，表示返回的是更新前的文档还是更新后的文档

其中，update和remove有且只能由一个

该命令的限制：当匹配不到文档时会产生错误，而且每次只能处理一个文档。

> db.runCommand({"findAndModify":"math","query":{"count":29},"sort":{"count":1},"update":{$in

c:{"count":2}},"new":true})

{

"lastErrorObject" : {

"updatedExisting" : true,

"n" : 1

},

"value" : {

"\_id" : ObjectId("58024856b58a9eca0cf5c9ac"),

"count" : 31

},

"ok" : 1

}

安全版本：

> db.runCommand({"getLastError":1})

{

"connectionId" : 10,

"n" : 0,

"syncMillis" : 0,

"writtenTo" : null,

"err" : null,

"ok" : 1

}

安全版本会有对执行的结果返回信息，而非安全模式可能没有返回信息

1. 查询

Find查询：

1. find的第一个参数指定查询的条件，也是一个文档==》{}

> db.test.find({"url":"www.baidu.com"})

{ "\_id" : ObjectId("5802258f5a808d2420e8f90e"), "url" : "www.baidu.com", "pageviews" : 15 }

{ "\_id" : ObjectId("58024b8e5a808d2420e8f913"), "url" : "www.baidu.com", "pageviews" : 2 }

1. 当不指定参数或者指定为{}时，则返回集合中的所有文档

> db.test.find()

{ "\_id" : ObjectId("5802258f5a808d2420e8f90e"), "url" : "www.baidu.com", "pageviews" : 15 }

{ "\_id" : ObjectId("5802283f5a808d2420e8f90f"), "name" : "joe", "sex" : "male", "location" :

"wisconsin", "favorite book" : [ "book1", "book2" ] }

{ "\_id" : ObjectId("58024b8e5a808d2420e8f913"), "url" : "www.baidu.com", "pageviews" : 2 }

> db.test.find({})

{ "\_id" : ObjectId("5802258f5a808d2420e8f90e"), "url" : "www.baidu.com", "pageviews" : 15 }

{ "\_id" : ObjectId("5802283f5a808d2420e8f90f"), "name" : "joe", "sex" : "male", "location" :

"wisconsin", "favorite book" : [ "book1", "book2" ] }

{ "\_id" : ObjectId("58024b8e5a808d2420e8f913"), "url" : "www.baidu.com", "pageviews" : 2 }

1. 第二个参数指定返回的键：节省传输的数据量，节省客户端解码文档的时间和内存消耗

> db.blog.posts.find()

{ "\_id" : ObjectId("58022b485a808d2420e8f910"), "title" : "a blog post", "content" : "...", "

author" : { "name" : "joe schmoe", "email" : "joe@example.com" }, "comments" : [ "good post."

, "can be added in.", "a", "b", "good post." ] }

{ "\_id" : ObjectId("58023ca55a808d2420e8f912"), "title" : "数组下标测试", "content" : "..", "

comments" : [ { "comment" : "good post", "author" : "jim", "votes" : 2 }, { "comment" : "i th

ought it was too short", "author" : "clarie", "votes" : 3 }, { "comment" : "free watches", "a

uthor" : "alice", "votes" : -1 } ] }

> db.blog.posts.find({},{"title":1,"content":1})//指定返回键

{ "\_id" : ObjectId("58022b485a808d2420e8f910"), "title" : "a blog post", "content" : "..." }

{ "\_id" : ObjectId("58023ca55a808d2420e8f912"), "title" : "数组下标测试", "content" : ".." }

> db.blog.posts.find({},{"content":0})//不返回content

{ "\_id" : ObjectId("58022b485a808d2420e8f910"), "title" : "a blog post", "author" : { "name"

: "joe schmoe", "email" : "joe@example.com" }, "comments" : [ "good post.", "can be added in.

", "a", "b", "good post." ] }

{ "\_id" : ObjectId("58023ca55a808d2420e8f912"), "title" : "数组下标测试", "comments" : [ { "c

omment" : "good post", "author" : "jim", "votes" : 2 }, { "comment" : "i thought it was too s

hort", "author" : "clarie", "votes" : 3 }, { "comment" : "free watches", "author" : "alice",

"votes" : -1 } ] }

4.2 查询条件

$lt ==> <

$lte ==> <=

$gt ==> >

$gte ==> >=

> db.math.find({"count":{$lt:20}})

{ "\_id" : ObjectId("58025a7f5a808d2420e8f914"), "count" : 1 }

{ "\_id" : ObjectId("58025a825a808d2420e8f915"), "count" : 12 }

{ "\_id" : ObjectId("58025a855a808d2420e8f916"), "count" : 14 }

{ "\_id" : ObjectId("58025a8a5a808d2420e8f918"), "count" : 13 }

{ "\_id" : ObjectId("58025a905a808d2420e8f91a"), "count" : 15 }

{ "\_id" : ObjectId("58025a995a808d2420e8f91d"), "count" : 12 }

{ "\_id" : ObjectId("58025aa05a808d2420e8f91f"), "count" : 19 }

{ "\_id" : ObjectId("58025aa35a808d2420e8f920"), "count" : 9 }

> db.math.find({"count":{$lt:80,$gt:20}})

{ "\_id" : ObjectId("58024856b58a9eca0cf5c9ac"), "count" : 31 }

{ "\_id" : ObjectId("58025a935a808d2420e8f91b"), "count" : 35 }

{ "\_id" : ObjectId("58025a965a808d2420e8f91c"), "count" : 25 }

{ "\_id" : ObjectId("58025a9d5a808d2420e8f91e"), "count" : 23 }

不能条件$ne

> db.blog.posts.find({})

{ "\_id" : ObjectId("58022b485a808d2420e8f910"), "title" : "a blog post", "content" : "...", "

author" : { "name" : "joe schmoe", "email" : "joe@example.com" }, "comments" : [ "good post."

, "can be added in.", "a", "b", "good post." ] }

{ "\_id" : ObjectId("58023ca55a808d2420e8f912"), "title" : "数组下标测试", "content" : "..", "

comments" : [ { "comment" : "good post", "author" : "jim", "votes" : 2 }, { "comment" : "i th

ought it was too short", "author" : "clarie", "votes" : 3 }, { "comment" : "free watches", "a

uthor" : "alice", "votes" : -1 } ] }

> db.blog.posts.find({"title":{$ne:"a blog post"}})

{ "\_id" : ObjectId("58023ca55a808d2420e8f912"), "title" : "数组下标测试", "content" : "..", "

comments" : [ { "comment" : "good post", "author" : "jim", "votes" : 2 }, { "comment" : "i th

ought it was too short", "author" : "clarie", "votes" : 3 }, { "comment" : "free watches", "a

uthor" : "alice", "votes" : -1 } ] }

or查询

$in 和$or

$in要加一个条件数组

> db.math.find({"count":{$in:[12,13,23,25,123]}})

{ "\_id" : ObjectId("58025a825a808d2420e8f915"), "count" : 12 }

{ "\_id" : ObjectId("58025a875a808d2420e8f917"), "count" : 123 }

{ "\_id" : ObjectId("58025a8a5a808d2420e8f918"), "count" : 13 }

{ "\_id" : ObjectId("58025a965a808d2420e8f91c"), "count" : 25 }

{ "\_id" : ObjectId("58025a995a808d2420e8f91d"), "count" : 12 }

{ "\_id" : ObjectId("58025a9d5a808d2420e8f91e"), "count" : 23 }

$nin排除掉数组中的元素

> db.math.find({"count":{$nin:[12,123,13,23,25,1213]}})

{ "\_id" : ObjectId("58024856b58a9eca0cf5c9ac"), "count" : 31 }

{ "\_id" : ObjectId("58025a7f5a808d2420e8f914"), "count" : 1 }

{ "\_id" : ObjectId("58025a855a808d2420e8f916"), "count" : 14 }

{ "\_id" : ObjectId("58025a8d5a808d2420e8f919"), "count" : 1345 }

{ "\_id" : ObjectId("58025a905a808d2420e8f91a"), "count" : 15 }

{ "\_id" : ObjectId("58025a935a808d2420e8f91b"), "count" : 35 }

{ "\_id" : ObjectId("58025aa05a808d2420e8f91f"), "count" : 19 }

{ "\_id" : ObjectId("58025aa35a808d2420e8f920"), "count" : 9 }

$or查询不同字段或的匹配

> db.test.find()

{ "\_id" : ObjectId("5802258f5a808d2420e8f90e"), "url" : "www.baidu.com", "pageviews" : 15 }

{ "\_id" : ObjectId("5802283f5a808d2420e8f90f"), "name" : "joe", "sex" : "male", "location" :

"wisconsin", "favorite book" : [ "book1", "book2" ] }

{ "\_id" : ObjectId("58024b8e5a808d2420e8f913"), "url" : "www.baidu.com", "pageviews" : 2 }

> db.test.find({$or:[{"name":"joe"},{"pageviews":2}]})

{ "\_id" : ObjectId("5802283f5a808d2420e8f90f"), "name" : "joe", "sex" : "male", "location" :

"wisconsin", "favorite book" : [ "book1", "book2" ] }

{ "\_id" : ObjectId("58024b8e5a808d2420e8f913"), "url" : "www.baidu.com", "pageviews" : 2 }

$not是元条件句，可以用在任何其他条件之上

特定于类型的查询：

Null:

null可以匹配它自身，而且还能匹配”不存在的”，因此匹配还会返回缺少这个键的所有文档

> db.test.find({"name":null})

{ "\_id" : ObjectId("5802258f5a808d2420e8f90e"), "url" : "www.baidu.com", "pageviews" : 15 }

{ "\_id" : ObjectId("58024b8e5a808d2420e8f913"), "url" : "www.baidu.com", "pageviews" : 2 }

因此，如果想要匹配键值为null的文档，既要检查键的值是null还要确保文档中存在该键。

> db.test.find({"url":{$in:[null],"$exists":true}})

{ "\_id" : ObjectId("580269d05a808d2420e8f921"), "url" : null, "pageviews" : 12 }

$in的数组中可以是不同类型

4.3.2 正则表达式

使用正则时表达式不用加””

> db.blog.posts.find()

{ "\_id" : ObjectId("58022b485a808d2420e8f910"), "title" : "a blog post", "content" : "...", "author" : { "name" : "joe schmoe", "email" : "joe@example.com" }, "comments" : [ "good post.", "can be added in.", "a", "b", "good post." ] }

{ "\_id" : ObjectId("58023ca55a808d2420e8f912"), "title" : "blog", "content" : "..", "comments" : [ { "comment" : "good post", "author" : "jim", "votes" : 2 }, { "comment" : "i thought it was too short", "author" : "clarie", "votes" : 3 }, { "comment" : "free watches", "author" : "alice", "votes" : -1 } ] }

> db.blog.posts.find({"title":/blog/i})

{ "\_id" : ObjectId("58022b485a808d2420e8f910"), "title" : "a blog post", "content" : "...", "author" : { "name" : "joe schmoe", "email" : "joe@example.com" }, "comments" : [ "good post.", "can be added in.", "a", "b", "good post." ] }

{ "\_id" : ObjectId("58023ca55a808d2420e8f912"), "title" : "blog", "content" : "..", "comments" : [ { "comment" : "good post", "author" : "jim", "votes" : 2 }, { "comment" : "i thought it was too short", "author" : "clarie", "votes" : 3 }, { "comment" : "free watches", "author" : "alice", "votes" : -1 } ] }

Mongodb支持perl兼容的所有正则表达式

4.3.3 查询数组

> db.food.insert({"fruit":["apple","banana","peach"]})

WriteResult({ "nInserted" : 1 })

> db.food.find()

{ "\_id" : ObjectId("58026d015a808d2420e8f922"), "fruit" : [ "apple", "banana", "peach" ] }

> db.food.find({"fruit":"banana"})//查询时可以当做键和值来查询

{ "\_id" : ObjectId("58026d015a808d2420e8f922"), "fruit" : [ "apple", "banana", "peach" ] }

1. $all

可以通过多个元素来匹配数组

> db.food.insert({"\_id":3,"fruit":["cherry","banana","aple"]})

WriteResult({ "nInserted" : 1 })

> db.food.find()

{ "\_id" : ObjectId("58026d015a808d2420e8f922"), "fruit" : [ "apple", "banana", "peach" ] }

{ "\_id" : 1, "fruit" : [ "apple", "banana", "peach" ] }

{ "\_id" : 2, "fruit" : [ "apple", "kumquat", "orange" ] }

{ "\_id" : 3, "fruit" : [ "cherry", "banana", "aple" ] }

> db.food.find({"fruit":{$all:["apple","banana"]}})//查询出都包含apple和banana的文档

{ "\_id" : ObjectId("58026d015a808d2420e8f922"), "fruit" : [ "apple", "banana", "peach" ] }

{ "\_id" : 1, "fruit" : [ "apple", "banana", "peach" ] }

可以使用精确匹配

> db.food.find({"fruit":{$all:["apple","banana"]}})

{ "\_id" : ObjectId("58026d015a808d2420e8f922"), "fruit" : [ "apple", "banana", "peach" ] }

{ "\_id" : 1, "fruit" : [ "apple", "banana", "peach" ] }

> db.food.insert({"fruit":["apple","banana"]})

WriteResult({ "nInserted" : 1 })

> db.food.find({"fruit":["apple","banana"]})//精确匹配

{ "\_id" : ObjectId("58026fed5a808d2420e8f923"), "fruit" : [ "apple", "banana" ] }

要想匹配数组指定位置的元素，则需要使用key.index语法指定下标

> db.food.find({"fruit.0":"apple"})

{ "\_id" : ObjectId("58026d015a808d2420e8f922"), "fruit" : [ "apple", "banana", "peach" ] }

{ "\_id" : 1, "fruit" : [ "apple", "banana", "peach" ] }

{ "\_id" : 2, "fruit" : [ "apple", "kumquat", "orange" ] }

{ "\_id" : ObjectId("58026fed5a808d2420e8f923"), "fruit" : [ "apple", "banana" ] }

1. $size

可以指定查询数组的长度

> db.food.find()

{ "\_id" : ObjectId("58026d015a808d2420e8f922"), "fruit" : [ "apple", "banana", "peach" ] }

{ "\_id" : 1, "fruit" : [ "apple", "banana", "peach" ] }

{ "\_id" : 2, "fruit" : [ "apple", "kumquat", "orange" ] }

{ "\_id" : 3, "fruit" : [ "cherry", "banana", "aple" ] }

{ "\_id" : ObjectId("58026fed5a808d2420e8f923"), "fruit" : [ "apple", "banana" ] }

> db.food.find({"fruit":{$size:2}})

{ "\_id" : ObjectId("58026fed5a808d2420e8f923"), "fruit" : [ "apple", "banana" ] }

1. $sliec操作符

返回数组的一个子集合

> db.food.find({},{"fruit":{$slice:2}})//其中fruit数组只返回前两个元素

{ "\_id" : ObjectId("58026d015a808d2420e8f922"), "fruit" : [ "apple", "banana" ] }

{ "\_id" : 1, "fruit" : [ "apple", "banana" ] }

{ "\_id" : 2, "fruit" : [ "apple", "kumquat" ] }

{ "\_id" : 3, "fruit" : [ "cherry", "banana" ] }

{ "\_id" : ObjectId("58026fed5a808d2420e8f923"), "fruit" : [ "apple", "banana" ] }

> db.food.find({},{"fruit":{$slice:-2}})//其中fruit数组只返回后两个元素

{ "\_id" : ObjectId("58026d015a808d2420e8f922"), "fruit" : [ "banana", "peach" ] }

{ "\_id" : 1, "fruit" : [ "banana", "peach" ] }

{ "\_id" : 2, "fruit" : [ "kumquat", "orange" ] }

{ "\_id" : 3, "fruit" : [ "banana", "aple" ] }

{ "\_id" : ObjectId("58026fed5a808d2420e8f923"), "fruit" : [ "apple", "banana" ] }

> db.food.find({},{"fruit":{$slice:[1,1]}})//返回指定偏移值后几个元素

{ "\_id" : ObjectId("58026d015a808d2420e8f922"), "fruit" : [ "banana" ] }

{ "\_id" : 1, "fruit" : [ "banana" ] }

{ "\_id" : 2, "fruit" : [ "kumquat" ] }

{ "\_id" : 3, "fruit" : [ "banana" ] }

{ "\_id" : ObjectId("58026fed5a808d2420e8f923"), "fruit" : [ "banana" ] }

4.3.4 查询内嵌文档

1. 内嵌文档匹配

> db.people.find()

{ "\_id" : ObjectId("5802b3ba5a808d2420e8f924"), "name" : { "first" : "joe", "last" : "schmoe" }, "age" : 4

5 }

> db.people.find({"name":{"first":"joe","last":"schmoe"}})

{ "\_id" : ObjectId("5802b3ba5a808d2420e8f924"), "name" : { "first" : "joe", "last" : "schmoe" }, "age" : 4

5 }

内嵌文档查询匹配要求整个文档完全匹配，否则就就用点表示法

> db.people.find()

{ "\_id" : ObjectId("5802b3ba5a808d2420e8f924"), "name" : { "first" : "joe", "last" : "schmoe" }, "age" : 4

5 }

> db.people.find({"name":{"first":"joe"}})//last没有匹配

> db.people.find({"name":{"first":"joe","last":"schmoe"}})

{ "\_id" : ObjectId("5802b3ba5a808d2420e8f924"), "name" : { "first" : "joe", "last" : "schmoe" }, "age" : 4

5 }

1. 使用点表示法查询内嵌的键

> db.people.find({"name.first":"joe","name.last":"schmoe"})

{ "\_id" : ObjectId("5802b3ba5a808d2420e8f924"), "name" : { "first" : "joe", "last" : "schmoe" }, "age" : 4

5 }

> db.runCommand({"getLastError":1})

{

"connectionId" : 10,

"n" : 0,

"syncMillis" : 0,

"writtenTo" : null,

"err" : null,

"ok" : 1

}

查询文档可以包含点，来表达”深入内嵌文档内部”的意思，因此待插入的文档的键不能包含点

当需要在键中插入点时，一种解决办法为在插入之前或之后执行一个全局替换，将点”.”替换成一个其他字符

要正确的指定一组条件，而不用指定每个键，要使用$elemMatch,该方法仅当需要对一个内嵌文档的多个键操作时才会用到

> db.blog.posts.find({"comments":{$elemMatch:{"author":"jim","votes":{$gte:2}}}})

{ "\_id" : ObjectId("58023ca55a808d2420e8f912"), "title" : "blog", "content" : "..", "comments" : [ { "comm

ent" : "good post", "author" : "jim", "votes" : 2 }, { "comment" : "i thought it was too short", "author"

: "clarie", "votes" : 3 }, { "comment" : "free watches", "author" : "alice", "votes" : -1 } ] }

4.4 $where查询

$where子句，用他可以执行任意javascript作为查询的一部分

> db.foo.find()

{ "\_id" : ObjectId("5802ba3c5a808d2420e8f925"), "apple" : 1, "banana" : 6, "peach

" : 3 }

{ "\_id" : ObjectId("5802ba5e5a808d2420e8f926"), "apple" : 8, "spinach" : 4, "wate

relon" : 4 }

> db.foo.find({$where:"this.x+this.y==7"})

> db.foo.find({$where:function(){return this.x+this.y==10;}})

不必要时要避免使用$where语句,查询速度慢（每个文档都转换为javascript对象，也不能利用索引）

4.5 游标

创建集合

> for(i=0;i<100;i++){

... db.c.insert({x:i});

... }

WriteResult({ "nInserted" : 1 })

> db.c.find()

{ "\_id" : ObjectId("5802bd2e5a808d2420e8f927"), "x" : 0 }

{ "\_id" : ObjectId("5802bd2f5a808d2420e8f928"), "x" : 1 }

...

迭代游标hasNext判断，next返回

> var cursor=db.c.find()

> cursor.hasNext()

true

> cursor.next()

{ "\_id" : ObjectId("5802bd2e5a808d2420e8f927"), "x" : 0 }

使用游标的迭代器接口;

> cursor.forEach(function(x){

... print(x.x);

... })

1

2

3

当调用find的时候，shell并不立即查询数据库，而是等待真正开始要求获得结果的时候才发送查询，这样在执行之前可以给查询附加额外的选项，几乎所有游标对象的方法都返回游标本身，这样就可以按任意顺序组成方法链。

> var cursor=db.c.find().skip(30).sort({"x":1}).limit(10)//这时并没有开始到数据库查询

> cursor.hasNext()

4.5.1 limit 、skip和sort

限制查询返回的结果数量，或者忽略一定数量的结果并排序，这些选项都要在查询被派发到服务器之前添加。

限制数量上限：limit函数

> db.c.find().limit(2)

{ "\_id" : ObjectId("5802bd2e5a808d2420e8f927"), "x" : 0 }

{ "\_id" : ObjectId("5802bd2f5a808d2420e8f928"), "x" : 1 }

忽略一定数量的文档：skip函数，注意：如果集合里面能匹配文档少于忽略的个数，则不会返回任何的文档。

排序函数：sort，参数为一组键值对，键对应排序的键名，键值为-1则为降序，为1则为升序。如果定义了多个排序的键值对，则按照排序的顺序依次排序

> db.c.find().sort({"x":-1})//按x降序排列

{ "\_id" : ObjectId("5802bd2f5a808d2420e8f98a"), "x" : 99 }

{ "\_id" : ObjectId("5802bd2f5a808d2420e8f989"), "x" : 98 }

...

> db.blog.posts.find().sort({"comments":1})

{ "\_id" : ObjectId("58022b485a808d2420e8f910"), "title" : "a blog post", "content" : "...", "author" : { "nam

e" : "joe schmoe", "email" : "joe@example.com" }, "comments" : [ "good post.", "can be added in.", "a", "b",

"good post." ] }

{ "\_id" : ObjectId("58023ca55a808d2420e8f912"), "title" : "blog", "content" : "..", "comments" : [ { "comment

" : "good post", "author" : "jim", "votes" : 2 }, { "comment" : "i thought it was too short", "author" : "cla

rie", "votes" : 3 }, { "comment" : "free watches", "author" : "alice", "votes" : -1 } ] }

> db.food.find()

{ "\_id" : ObjectId("58026d015a808d2420e8f922"), "fruit" : [ "apple", "banana", "peach" ] }

{ "\_id" : 1, "fruit" : [ "apple", "banana", "peach" ] }

{ "\_id" : 2, "fruit" : [ "apple", "kumquat", "orange" ] }

{ "\_id" : 3, "fruit" : [ "cherry", "banana", "aple" ] }

{ "\_id" : ObjectId("58026fed5a808d2420e8f923"), "fruit" : [ "apple", "banana" ] }

> db.food.find().sort({"\_id":1,"fruit":-1})//先按\_id升序在按fruit降序

{ "\_id" : 1, "fruit" : [ "apple", "banana", "peach" ] }

{ "\_id" : 2, "fruit" : [ "apple", "kumquat", "orange" ] }

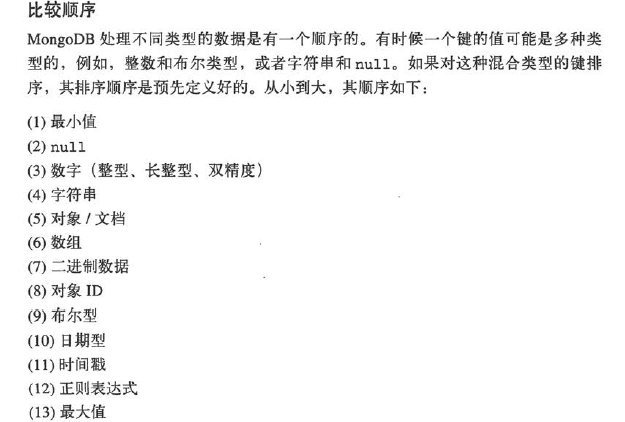
{ "\_id" : 3, "fruit" : [ "cherry", "banana", "aple" ] }

{ "\_id" : ObjectId("58026d015a808d2420e8f922"), "fruit" : [ "apple", "banana", "peach" ] }

{ "\_id" : ObjectId("58026fed5a808d2420e8f923"), "fruit" : [ "apple", "banana" ] }

这三个方法对于分页来说会非常有用

Mongodb对于一个键有很多类型的数值时有一个预先定义好的顺序



4.5.2 避免使用skip略过大量的结果

当数据链很大时，使用skip略过数据就会产生性能问题（几乎每个数据库都会遇到这个问题）

解决办法：

1. 不使用skip对结果进行分页
2. 随机选取文档：添加随机键

4.5.3 高级查询选项

1. 普通查询:

> var cursor=db.foo.find()

1. 包装查询

Var cursor=db.foo.find({“foo”:”bar”}).sort({“x”:1})

实际情况不是将{“foo”:”bar”}作为查询条件直接发送给数据库，而是将查询包装在一个更大的文档中，shell会把查询语句从{“foo”:”bar”}转换为{$query:{“foo”:”bar”},$orderby:{“x”:1}}

4.5.4 获取一致结果

解决办法：使用查询快照

游标有作用域

1. 索引

加速查询速度

创建索引：

> db.c.ensureIndex({"x":1})

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

返回的文档时可以拿到数据的:

> db.c.ensureIndex({"x":1}).ok

1

对某个键创建的索引会加速对该键的查询，而对于其他的查询则没有帮助，即便是查询中包含了被索引的键，因此通常来说，一定要创建查询中用到的所有的索引。

$ensureIndex的文档的形式和sort的一样，值为1或-1的键，表示创建索引的方向（1为升序，-1为降序）。若索引只有一个键则方向无关紧要。

5.1.2 为内嵌文档中的键创建索引，与为普通的键创建索引没有什么区别

> db.people.find()

{ "\_id" : ObjectId("5802b3ba5a808d2420e8f924"), "name" : { "first" : "joe", "last"

: "schmoe" }, "age" : 45 }

> db.people.ensureIndex({"name.first":1})//使用点操作符

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

为内嵌文档创建的索引也可以和普通键创建的索引组合成符合索引

5.1.3 为排序创建索引

对非索引字段的排序是在内存中进行的

5.1.4 自定义索引名称

> db.people.ensureIndex({"name.last":1},{"name":"lastname\_index"})//新的索引名成为lastname\_index

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 2,

"numIndexesAfter" : 3,

"ok" : 1

}

5.2 唯一索引

\_Id的索引是唯一索引而且不能删除。

如果没有对应的键，索引会将其作为null存储。

5.2.1 消除重复

有些时候，可能希望将所有包含重复值的文档都删掉，dropDups选项就可以保留发现的第一个文档，而删除接下来的有重复值得文档。

5.2.2 复合唯一索引

创建复合索引：

> db.people.ensureIndex({"name.first":1,"name.last":-1})

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 3,

"numIndexesAfter" : 4,

"ok" : 1

}

5.3 使用explain和hint

explain函数，对游标调用该方法，就可以得到查询细节，会返回一个文档

> db.people.find().explain()

{

"queryPlanner" : {

"plannerVersion" : 1,

"namespace" : "test.people",

"indexFilterSet" : false,

"parsedQuery" : {

"$and" : [ ]

},

"winningPlan" : {

"stage" : "COLLSCAN",

"filter" : {

"$and" : [ ]

},

"direction" : "forward"

},

"rejectedPlans" : [ ]

},

"serverInfo" : {

"host" : "CHINA-20160318B",

"port" : 27017,

"version" : "3.2.9",

"gitVersion" : "22ec9e93b40c85fc7cae7d56e7d6a02fd811088c"

},

"ok" : 1

}

> db.c.find({"x":{$lte:10}}).explain()

{

"queryPlanner" : {

"plannerVersion" : 1,

"namespace" : "test.c",

"indexFilterSet" : false,

"parsedQuery" : {

"x" : {

"$lte" : 10

}

},

"winningPlan" : {

"stage" : "FETCH",

"inputStage" : {

"stage" : "IXSCAN",

"keyPattern" : {

"x" : 1

},

"indexName" : "x\_1",//使用了索引

"isMultiKey" : false,

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 1,

"direction" : "forward",

"indexBounds" : {

"x" : [

"[-1.#INF, 10.0]"

]

}

}

},

"rejectedPlans" : [ ]

},

"serverInfo" : {

"host" : "CHINA-20160318B",

"port" : 27017,

"version" : "3.2.9",

"gitVersion" : "22ec9e93b40c85fc7cae7d56e7d6a02fd811088c"

},

"ok" : 1

}

索引查询：

> db.test.ensureIndex({"url":1})

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

> db.test.find({"url":"www.baidu.com","pageviews":{$gte:1}}).explain()

{

"queryPlanner" : {

"plannerVersion" : 1,

"namespace" : "test.test",

"indexFilterSet" : false,

"parsedQuery" : {

"$and" : [

{

"url" : {

"$eq" : "www.baidu.com"

}

},

{

"pageviews" : {

"$gte" : 1

}

}

]

},

"winningPlan" : {

"stage" : "FETCH",

"filter" : {

"pageviews" : {

"$gte" : 1

}

},

"inputStage" : {

"stage" : "IXSCAN",

"keyPattern" : {

"url" : 1

},

"indexName" : "url\_1",

"isMultiKey" : false,

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 1,

"direction" : "forward",

"indexBounds" : {

"url" : [

"[\"www.baidu.com\", \"www.baidu.co

m\"]"

]

}

}

},

"rejectedPlans" : [ ]

},

"serverInfo" : {

"host" : "CHINA-20160318B",

"port" : 27017,

"version" : "3.2.9",

"gitVersion" : "22ec9e93b40c85fc7cae7d56e7d6a02fd811088c"

},

"ok" : 1

}

强制mongodb使用某一索引：hint函数

> db.test.find({"url":"www.baidu.com"}).hint({"url":1})//强制使用这个索引

{ "\_id" : ObjectId("5802258f5a808d2420e8f90e"), "url" : "www.baidu.com", "pageviews

" : 15 }

{ "\_id" : ObjectId("58024b8e5a808d2420e8f913"), "url" : "www.baidu.com", "pageviews

" : 2 }

删除指定名字的索引：

> db.runCommand({"dropIndexes":"foo","index":"url\_1"})

{

"nIndexesWas" : 1,

"ok" : 0,//没有找到该索引则执行失败

"errmsg" : "index not found with name [url\_1]",

"code" : 27

}

> db.runCommand({"dropIndexes":"people","index":"name.first\_1"})

{ "nIndexesWas" : 4, "ok" : 1 }//删除索引成功

> db.runCommand({"dropIndexes":"people","index":"\*"})//删除所有索引

{

"nIndexesWas" : 3,

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

"ok" : 1

}

5.5 地理空间索引

Mongodb为坐标平面查询提供了专门的索引，称作地理空间索引

创建地理空间索引：ensureIndex函数，值为2d

创建测试数据：

> for(i=0;i<100;i++){ db.map.insert({gps:{x:2\*i+1,y:3\*i+2}}); }

WriteResult({ "nInserted" : 1 })

> db.map.find()

{ "\_id" : ObjectId("580340405a808d2420e8f9ef"), "gps" : { "x" : 1, "y" : 2 } }

{ "\_id" : ObjectId("580340405a808d2420e8f9f0"), "gps" : { "x" : 3, "y" : 5 } }...

为map集合创建地理空间索引：要创建地理空间索引的键必须是某种形式的一对值：一个包含两个元素的数组或是包含两个键的内嵌文档

> db.map.ensureIndex({"gps":"2d"})

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

地理空间查询以两种方式进行：即普通查询和使用数据库命令。

1. 普通查询与以往的查询差别不大，但是增加了$near方式，需要两个目标值的数组作为参数

> db.map.find({"gps":{$near:[20,100]}})

{ "\_id" : ObjectId("580340405a808d2420e8fa09"), "gps" : { "x" : 53, "y" : 80 } }

{ "\_id" : ObjectId("580340405a808d2420e8fa08"), "gps" : { "x" : 51, "y" : 77 } }

{ "\_id" : ObjectId("580340405a808d2420e8fa0a"), "gps" : { "x" : 55, "y" : 83 } }

{ "\_id" : ObjectId("580340405a808d2420e8fa07"), "gps" : { "x" : 49, "y" : 74 } }

...

这会由远及近的方式返回所有文档

1. 也可以使用geoNear来完成相同的操作

> db.runCommand({geoNear:"map",near:[50,100],num:10})

{

"waitedMS" : NumberLong(0),

"results" : [

{

"dis" : 13.601470508735444,

"obj" : {

"\_id" : ObjectId("580340405a808d2420e8fa0d"),

"gps" : {

"x" : 61,

"y" : 92

}

}

},

{

"dis" : 13.92838827718412,

"obj" : {

"\_id" : ObjectId("580340405a808d2420e8fa0e"),

"gps" : {

"x" : 63,

"y" : 95

}

}

},

{

"dis" : 14.212670403551895,

"obj" : {

"\_id" : ObjectId("580340405a808d2420e8fa0c"),

"gps" : {

"x" : 59,

"y" : 89

}

}

},

{

"dis" : 15.132745950421556,

"obj" : {

"\_id" : ObjectId("580340405a808d2420e8fa0f"),

"gps" : {

"x" : 65,

"y" : 98

}

}

},

{

"dis" : 15.652475842498529,

"obj" : {

"\_id" : ObjectId("580340405a808d2420e8fa0b"),

"gps" : {

"x" : 57,

"y" : 86

}

}

},

{

"dis" : 17.029386365926403,

"obj" : {

"\_id" : ObjectId("580340405a808d2420e8fa10"),

"gps" : {

"x" : 67,

"y" : 101

}

}

},

{

"dis" : 17.72004514666935,

"obj" : {

"\_id" : ObjectId("580340405a808d2420e8fa0a"),

"gps" : {

"x" : 55,

"y" : 83

}

}

},

{

"dis" : 19.4164878389476,

"obj" : {

"\_id" : ObjectId("580340405a808d2420e8fa11"),

"gps" : {

"x" : 69,

"y" : 104

}

}

},

{

"dis" : 20.223748416156685,

"obj" : {

"\_id" : ObjectId("580340405a808d2420e8fa09"),

"gps" : {

"x" : 53,

"y" : 80

}

}

},

{

"dis" : 22.135943621178654,

"obj" : {

"\_id" : ObjectId("580340405a808d2420e8fa12"),

"gps" : {

"x" : 71,

"y" : 107

}

}

}

],

"stats" : {

"nscanned" : 54,

"objectsLoaded" : 52,

"avgDistance" : 16.905336237127024,

"maxDistance" : 22.135943621178654,

"time" : 45

},

"ok" : 1

}

Mongodb可以找到指定形状内的文档

5.5.1 复合地理空间索引

普通索引和地理空间索引的组合可以查询类似某地周围所有咖啡馆或者披萨店

> db.location.ensureIndex({"location":"2d","desc":1},{"name":"地理空间复合索引"})

{

"createdCollectionAutomatically" : true,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

> db.location.find({"location":{$near:[10,20]},"desc":/.\*coffee\*./i})

1. 聚合

6.1 count

> db.map.count()

60

不论文档多大，都会很快返回总的文档数量

也可以传递参数，限定条件

> db.map.count({"gps.x":{$lt:30},"gps.y":{$gt:20}})

8

6.2 distinct

用来找出给定键的所有不同的值，使用时不许指定集合和键

> db.food.find()

{ "\_id" : ObjectId("58026d015a808d2420e8f922"), "fruit" : [ "apple", "banana", "peach" ] }

{ "\_id" : 1, "fruit" : [ "apple", "banana", "peach" ] }

{ "\_id" : 2, "fruit" : [ "apple", "kumquat", "orange" ] }

{ "\_id" : 3, "fruit" : [ "cherry", "banana", "aple" ] }

{ "\_id" : ObjectId("58026fed5a808d2420e8f923"), "fruit" : [ "apple", "banana" ] }

> db.runCommand({"distinct":"food","key":"fruit"})

{

"waitedMS" : NumberLong(0),

"values" : [

"apple",

"banana",

"peach",

"kumquat",

"orange",

"aple",

"cherry"

],

"stats" : {

"n" : 5,

"nscanned" : 0,

"nscannedObjects" : 5,

"timems" : 0,

"planSummary" : "COLLSCAN"

},

"ok" : 1

}

> db.runCommand({"distinct":"map","key":"gps.x"})

{

"waitedMS" : NumberLong(0),

"values" : [

1,

3,

5,

7,

9,

11,

13,

15,

17,

19,

21,

23,

25,

27,

29,

31,

33,

35,

37,

39,

41,

43,

45,

47,

49,

51,

53,

55,

57,

59,

61,

63,

65,

67,

69,

71,

73,

75,

77,

79,

81,

83,

85,

87,

89,

91,

93,

95,

97,

99,

101,

103,

105,

107,

109,

111,

113,

115,

117,

119

],

"stats" : {

"n" : 60,

"nscanned" : 0,

"nscannedObjects" : 60,

"timems" : 0,

"planSummary" : "COLLSCAN"

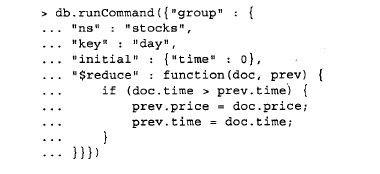
},

"ok" : 1

}

6.3 group

先选定分组所依据的键，而后mongodb就会将集合依据选定的键值的不同分成若干组，探后可以通过聚合每一组内的文档，产生一个结果文档。



未完

6.4 mapreduce

附录B MongoDB shell

启动shell时指定端口

Mongo.exe <http://www.127.0.0.1:27017>