Mongodb

Mongodb对于大数据量、高并发以及弱事物的互联网引用是个利器。

Mongodb内置水平扩展机制，从而可以处理百万甚至十亿级别的数据存储。

Mongodb不像关系型数据库，每张表都是预先定义的字段（也就限制了规则），而mongodb完全可将一个对象的所有属性放在一个文档里面，而不用受其他的什么限制。这样也使得mongodb的扩展非常的容易，不像关系型数据库，关系型数据库的在线模式更新不是一件容易的事情。

Mongodb的集合（类似于关系型数据库中的表）支持松散的模式，易于灵活调整，并且支持复杂的属性，可以建立索引作为查询条件，并可以直接对记录的某个字段进行原子性的改变。

mongodb支持当今主流的web应用的关键功能：索引、复制、分片和丰富的查询语法以及特别灵活的数据模型。当然也不失速度。

第一章

Mongodb是面向文档的数据库（所以mongodb没有模式即文档的键不会预先定义也不会固定不变），不必键所有数据都放到一个模子里面

。不是关系型数据库，这方便了它的可扩展性，某个文档的键值对可以随意添加或者删除。

Mongodb采用的面向文档的数据模型使其可以自动在多台服务器之间分隔数据。它还可以平衡集群的数据和负载（就像memcache开发的就是分布式的，自动在服务器之间选择存储；但是memcache是客户端决定的分布式，而mongodb是服务器自己实现分布式）以及自动重排文档。

memcache中是客户端来决定在哪台服务器存储数据，mongodb则是服务器自己决定的。两者一样，如果需要更大的容量只需要在集群中添加机器即可。

索引：

memcache支持通用辅助索引，能进行多种快速查询。也提供唯一的、复合的和地理空间索引能力。

mongodb可直接在服务端存储javascript的函数和值。

mongodb支持mapreduce和其他聚合工具。

mongodb还支持固定的大小的集合，即集合的大小是有上限的。

mongodb支持用一种容易使用的协议存储大型文件和文件的元数据。

与其他的关系型数据库不同的是，mongodb不支持联结查询和事务；这样设计是为了考虑扩展性。

Mongodb使用Mongodb传输协议（其他如http协议和rest协议需要更多的开销）作为与服务器交互的主要方式。

mongodb的默认存储引擎使用了内存映射文件，将内存的管理工作交给了操作系统去处理。

动态查询优化器会记住最高效的方式。总之，是在各个方面考虑了性能。

简便的管理：

如果主服务器挂掉了，mongodb会自动切换到备份服务器上，并且将备份服务器提升为活跃服务器。在分布式环境下，集群只需要知道有新增加的节点，就会自动集成和配置新的节点。

1. 入门

文档是基本的数据单元，类比RDMS的行，相似的集合相当于是表table。

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（它是更根据机器名、时间戳、进程id等生成的id），从这个id中可以得到创建这个文档的时间。另外这个id是可以自定义的而且自动添加了索引。

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。一般来讲mongodb会自动为插入的文档添加键为\_id，如果用户不自定义\_id的话就不会重复。

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（不像RDBMS里面，规则都在创建表的时候定义好了）：

{“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.authors对应两个集合，这里的blog不是必须的，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（貌似当前不是这样的，有待验证），因此数据库的名字有很多的限制。

集合由文档组成，多个集合可以组成数据库。一个mongodb实例可以承载多个数据库，它们之间可以视为完全独立的，而且每个数据库有各自独立的权限。不同的数据库放在在不同的文件中。

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

这个名字可以使用db.集合名来返回。

端口号是：27017

Mongodb自带javascript的shell，通过这个shell可以与mongodb实例进行交互，这里可运行任何的javascript程序，它是一个功能完备的javascript解释器，还可以充分利用javascript的标准库（如数字运算等）

> 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

2.5.2 Mongodb客户端

开始的时候shell会连接到mongodb的test数据库，并将这个数据库连接赋值给全局变量db

2.5.3 shell中的基本操作

与javascript中一样，文档的键可以不用双引号括起来，但值是字符串的时候必须要引起来。

1. 创建--C

> post={"title":"my blog post.",//使用局部变量post，命名空间就是 数据库.集合名

... "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:返回所有匹配的文档（默认一次性自动显示20个文档），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

> db.arr.remove({cc:"cc"})

WriteResult({ "nRemoved" : 1 })

如果不指定条件的删除则会删除该集合下的所有文档。

> db.arr.remove({}) //这将删除arr集合下的所有文档，但是集合arr并不会被删除

注意不指定参数是指传递一个空文档，而不是不添加参数，如果不添加任何的参数的话会报错。

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：

与json对象一样，键的双引号不是必须的。

Null:用来表示空值或者不存在的字段{“x”:null}

> db.ff.find({cc:null}) //这句话既能该查询出cc为null的文档，也能够查询出不含cc字段的文档。

布尔类型：true 和 false

{“x”:true}

shell中的数字类型都是64为浮点型，这是因为js只支持此类型。

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" } //返回的是字符串表示

> t={time:new Date()}

{ "time" : ISODate("2017-01-20T10:23:05.336Z") } //返回的是日期对象

Date()是构造函数，实际上返回的是日期的字符串表示，而new Date()则返回真正的日期对象。

shell中的日期显示时使用的是本地取的时区设置，而在数据中实际是以从标准纪元开始的毫秒的形式存储的。

有序对象：列表 栈 队列

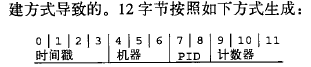
无序列表：集合

2.6.6 \_id和ObjectId

mongodb中存储的文档必须有一个\_id键（无论是mongodb添加的还是自己存储数据时指定的），这个键的值可以是任何类型的（自己指定，但是也不一定，数组就不可以）。默认是个ObjectId对象

mongodb从一开始就是用来作为分布式数据库，处理多个节点是一个核心要求，因此同样的集合可能存储在不同的服务器上，这样的话就不能像关系型数据库那样维护一个自增的字段，所以为了此要求加入了一个\_id的字段，即使一个集合的文档存储在不同的机器上面，一个集合中的文档的\_Id键也是唯一的，而且ObjectId在分片环境中容易生成的多。

生成方式：



如果插入的文档没有\_id键，系统就会自动帮你创建一个。可以由mongodb的服务器来做此事，当然通常在客户端有驱动生成（扩展应用层要必扩展数据数据库层容易的多）将事务交由客户端来处理，就减轻了客户端扩展的负担。

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

一次发送数十 数百乃至数千个文档会明显提高插入速度，这主要是因为tcp请求消息头的存在，用批量插入的话数据库就不用一遍又一遍的处理每个文档的这种信息了。

可以使用mongoimport从mysql等外部批量导入数据库。

3.1.2 插入：原理和作用

驱动程序会将数据转换为bson格式，数据库检查是否包含\_id键和文档大小是否超过4MB，然后不进行任何其他的验证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

三种类别：删除指定匹配模式的文档；删除集合中的所有文档但保留集合；删除整个集合。

> 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({})

删除数据是持久的，不能撤销也不能回复。

查看帮助：

Help

查看集合的命令帮助：db.mycoll.help()

3.3 更新文档

语法：

> db.arr.update

function (query, obj, upsert, multi) {

第一个参数代表查询文档；第二个参数代表修改器文档；第三个参数代表当查询文档不存在时是否将修改器文档插入集合；第四个参数代表是否对所有匹配项（如查询条件多个匹配）同时更新。

在mongo的shell中，操作bson可以跟操作json对象一样。

直接使用.操作符可以操作文档的属性。

> bson={}

{ }

> bson.name="abc"

abc

> bson.age="16"

16

> bson

{ "name" : "abc", "age" : "16" }

更新操作是原子的，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；修改器更新是原子的。

键的匹配都必须要使用双引号。

> 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}) //键名可以不用双引号括起来

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；键名可以不用双引号括起来

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，因此整个文档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. $push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."

]

}

使用$ne作为筛选匹配条件，$ne只能用在数组上：

> 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可以从数组的任何数组的一端删除一个元素，

> 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"

]

}

$pull可以删除数组元素的指定元素（只需要执行匹配的条件），$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，这个函数只能对单一的文档进行更改，多个匹配的情况下不会修改。

... )

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"

]

}

多个匹配的情况下：

> db.test.find({"title":"a"})

{ "\_id" : ObjectId("5809dcac7d98661ec9d4d0e1"), "title" : "a", "comments" : [ "a", "b", "c" ] }

{ "\_id" : ObjectId("5809dcb27d98661ec9d4d0e2"), "title" : "a", "comments" : [ "a1", "b1", "c1" ] }

{ "\_id" : ObjectId("5809dcb97d98661ec9d4d0e3"), "title" : "a", "comments" : [ "a2", "b2", "c2" ] }

{ "\_id" : ObjectId("5809dcc87d98661ec9d4d0e4"), "title" : "a", "comments" : [ "a3", "b3", "c3" ] }

> db.test.update({"title":"a"},{$pull:{"comments":"a"}})

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

> db.test.find({"title":"a"})

{ "\_id" : ObjectId("5809dcac7d98661ec9d4d0e1"), "title" : "a", "comments" : [ "b", "c" ] }

{ "\_id" : ObjectId("5809dcb27d98661ec9d4d0e2"), "title" : "a", "comments" : [ "a1", "b1", "c1" ] }

{ "\_id" : ObjectId("5809dcb97d98661ec9d4d0e3"), "title" : "a", "comments" : [ "a2", "b2", "c2" ] }

{ "\_id" : ObjectId("5809dcc87d98661ec9d4d0e4"), "title" : "a", "comments" : [ "a3", "b3", "c3" ] }

Pull不能嵌套替换：

> db.blog.update({"title":{$in:["a","b"]}},{$pull:{"comments.name":"a"}}) //这样并不能替换成功

WriteResult({

"nMatched" : 0,

"nUpserted" : 0,

"nModified" : 0,

"writeError" : {

"code" : 16837,

"errmsg" : "cannot use the part (comments of comments.name) to traverse the element ({comments: [ { name: \"a\", comment: \"a1\" }, { name: \"b\", comment: \"b1\" } ]})"

}

})

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—命令就是update,先查找，找到则更新，没有找到则创建,update命令的第三个参数为true则表示这是个upsert更新：

如果没有匹配的就连查询条件一起创建，否则就创建或者更新 update的条件。

> 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,

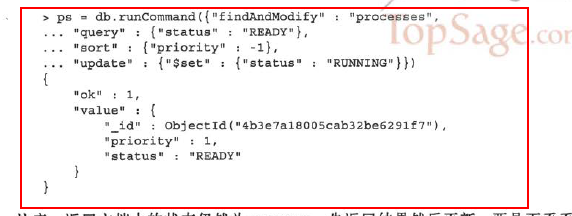
"err" : null,

"ok" : 1

}

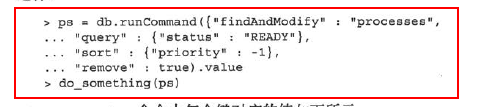
findAndModify：

1、查找并更新在一个操作内完成。



findAndModify的值为要查找的集合，query为查找的条件，sort为查找后的排列顺序，update为更新语句。

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；当给定值为true时返回更新后的文档。

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。

一般来讲，相比其他的操作findAndModify的操作可能更慢一些。

> 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

}

mongodb对插入 删除和更新的操作都是瞬间完成的，这是因为他们都不需要等待数据库的响应（这并不是说他们是异步的操作），客户端不会受到数据库执行的返回信息（除非该命令会返回执行信息），因此这些操作只会受到客户端发送速度和网络速度的制约。

mongodb的开发默认的是不安全的版本，安全模式在执行完了操作后会立即执行getLastError命令，来检查是否操作成功。但这会在每次操作时都返回数据，造成客户端的等待，降低了性能，但有利于发现服务器端的错误。

安全的代价就是性能。等待数据库响应本身的时间比只发送消息的时间多一个数量级。

开发阶段可以考虑使用安全模式。

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

为了提高效率建立了连接池：

语言的驱动使用了连接池：为了提高效率，这些驱动程序都和服务器建立了连接（一个连接池），并将请求分散到这些连接中去。

1. 查询

查询将会返回一个数据库游标，游标只有在你需要的时候才会惰性的批量返回文档。

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；remove的参数是{}时也会删除该集合下的所有文档。

> 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总是默认的返回，只有显示的指明\_Id不返回时才不会返回。

{ "\_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 }

不等条件$ne，在对数组元素的查询中表示不存在，该命令可以用于所有数据类型的数据：

> 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，即使只有一个元素，数组中的元素的类型可以不同。

> 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 }

$in数组中可以包含不同元素类型：

> db.arr.find({name:{$in:["a","c","e",23,{}]}})

{ "\_id" : ObjectId("5885a42b6e8915f9225d68cd"), "name" : "a" }

{ "\_id" : ObjectId("5885a4316e8915f9225d68cf"), "name" : "c" }

{ "\_id" : ObjectId("5885a4376e8915f9225d68d1"), "name" : "e" }

{ "\_id" : ObjectId("5885a4696e8915f9225d68d4"), "name" : 23 }

{ "\_id" : ObjectId("5885a4766e8915f9225d68d5"), "name" : { } }

$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 }

$or查询不同字段的匹配；$in可查询一个键的多个值，而$or可以查询多个键的匹配

$or接受一个包含所有可能条件的数组作为参数

> 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({$or:[{"name":"joe"},{"pageviews":2}]}) //$or接受一个数组作为参数

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

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

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

> db.arr.find({$or:[{name:"a"},{name:"c"}]})

{ "\_id" : ObjectId("5885a42b6e8915f9225d68cd"), "name" : "a" }

{ "\_id" : ObjectId("5885a4316e8915f9225d68cf"), "name" : "c" }

tips:使用普通的AND型查询时，总是想尽可能的用最少的条件来限定结果的范围，OR型的查询正相反，第一个条件尽可能匹配更多的文档，这样才是最为有效的。

$not是元条件句，可以用在任何其他条件之上，表示与模式条件不匹配的文档。

> db.num.find({id\_num:{$mod:[5,1]}})

{ "\_id" : ObjectId("5885a6f56e8915f9225d68d6"), "id\_num" : 1 }

{ "\_id" : ObjectId("5885a7016e8915f9225d68dc"), "id\_num" : 11 }

{ "\_id" : ObjectId("5885a7096e8915f9225d68df"), "id\_num" : 16 }

{ "\_id" : ObjectId("5885a70b6e8915f9225d68e0"), "id\_num" : 21 }

> db.num.find({id\_num:{$not:{$mod:[5,1]}}})

{ "\_id" : ObjectId("5885a6f86e8915f9225d68d7"), "id\_num" : 12 }

{ "\_id" : ObjectId("5885a6fa6e8915f9225d68d8"), "id\_num" : 2 }

{ "\_id" : ObjectId("5885a6fb6e8915f9225d68d9"), "id\_num" : 3 }

{ "\_id" : ObjectId("5885a6fe6e8915f9225d68da"), "id\_num" : 4 }

{ "\_id" : ObjectId("5885a6ff6e8915f9225d68db"), "id\_num" : 5 }

{ "\_id" : ObjectId("5885a7056e8915f9225d68dd"), "id\_num" : 12 }

{ "\_id" : ObjectId("5885a7076e8915f9225d68de"), "id\_num" : 14 }

{ "\_id" : ObjectId("5885a70e6e8915f9225d68e1"), "id\_num" : 222 }

{ "\_id" : ObjectId("5885a7116e8915f9225d68e2"), "id\_num" : 23 }

{ "\_id" : ObjectId("5885a7136e8915f9225d68e3"), "id\_num" : 25 }

4.2.4 条件语句的规则

更新修改器和查询条件语句书写规则：条件语句时内层文档的键，而修改器是外层文档的键。

修改器外层文档的键：

> db.num.update({id\_num:12},{$inc:{id\_num:5}})

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

条件语句内层文档的键，可以对一个键应用多个查询条件，所以条件语句的键写在了里面：

> db.num.find({id\_num:{$in:[222,23,11,17]}})

{ "\_id" : ObjectId("5885a6f86e8915f9225d68d7"), "id\_num" : 17 }

{ "\_id" : ObjectId("5885a7016e8915f9225d68dc"), "id\_num" : 11 }

{ "\_id" : ObjectId("5885a70e6e8915f9225d68e1"), "id\_num" : 222 }

{ "\_id" : ObjectId("5885a7116e8915f9225d68e2"), "id\_num" : 23 }

4.3 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”，因此这个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

mongodb可以为前缀正则表达式（比如/^joey/）查询创建索引，所以这种类型的查询会非常高效。

正则表达式也可以匹配自身，即查询出数据库中的正则表达式。

4.3.3 查询数组

查询数组中的元素也是非常容易的，数组绝大多数情况可以这样理解，每个元素都是整个键的值。

> 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（不过目前插入带有.的字符串时可行的）

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;}})

> db.num.find({$where:function(){

... for(var current in this){

... for(var other in this){

... if(current!=other && this[current]==this[other]){

... return true;

... }

... }

... }

... }

... })

{ "\_id" : ObjectId("5885a7056e8915f9225d68dd"), "id\_num" : 12, "id\_num2" : 12 }

使用$where进行过滤，如果函数返回true文档就作为结果的一部分被返回，如果为false则不然。

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）

除了函数以外，也可以使用字符串来指定$where的子查询：

> db.num.find({$where:"this.id\_num+this.id\_num2==15"})

{ "\_id" : ObjectId("5885a6fb6e8915f9225d68d9"), "id\_num" : 3, "id\_num2" : 12 }

> db.num.find({$where:"function(){ return this.id\_num+this.id\_num2==15; }" })

{ "\_id" : ObjectId("5885a6fb6e8915f9225d68d9"), "id\_num" : 3, "id\_num2" : 12 }

4.5 游标

mongodb使用游标来返回find的执行结果

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：count可以得到结果的总数，hasNext判断，next返回

> var cur=db.StarNode.find({Name:/^张/})

> cur.count() //有count函数

29619

> cur.hasNext() //判断是否还有为迭代的元素

true

> cur.next() //迭代下一个元素

> var cursor=db.c.find()

> cursor.hasNext()

true

> cursor.next()

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

游标实现了迭代器的接口，可以使用foreach循环游标得到结果：

> cursor.forEach(function(x){

... print(x.x);

... })

1

2

3

> cur.forEach(function(x){ print(x.id\_num); })

1

17

2

3

4

5

11

12

14

16

21

222

23

25

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" ] }

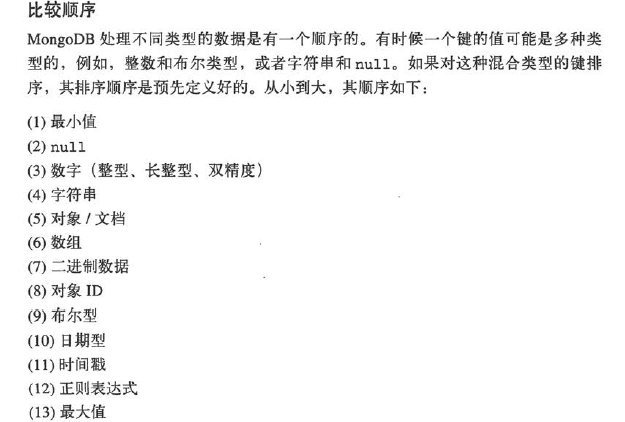
{ "\_id" : 3, "fruit" : [ "cherry", "banana", "aple" ] }

{ "\_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

不一致只在游标等待结果时集合内容改变的情况下发生。

解决办法：使用查询快照，$snapshot

客户端游标和服务器游标

游标消耗内存和其他资源

游标有作用域：如果客户端游标已经不在作用域中了，服务器端游标就会自动销毁释放资源。

1. 索引

加速查询

创建索引关键字:ensureIndex,可以同时给多个键创建索引。

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键的索引。

一次给多个键创建索引时实际创建的是一个索引，只是这个索引包含两个键。给多个键创建索引时需要考虑索引的方向。

$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。

一般来说，如果索引包含N个键，则对于前几个键的查询都会有帮助，比如有个索引{a:1,b:1,c:1….,z:1},是上是有了{a:1},{a:1,b:1},{a:1,b:1,c:1}等的索引，但是使用{b:1},{a:1,c:1}等索引的查询则不会被优化，只有使用索引前部的查询才能使用该索引。

mongodb的查询优化器会重排查询项的顺序，因此全部利用索引的时候键的顺序是无关紧要的，如查询{x:foo,y:bar}的时候，如果已经有了{y:1,x:1}的索引，mongodb会自动找到并利用此索引。

创建索引的缺点是每次插入、更新和删除时都会产生额外的开销。

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复合索引

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

所以做无索引排序是有限制的，就是不能在内存中做T级别数据的排序。

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

唯一索引可以确保集合的每一个文档的指定键都有唯一值。

> db.num.ensureIndex({id\_num:true},{unique:true})

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

insert并不检查文档是否插入过了

\_Id的索引是唯一索引而且不能删除。

如果没有对应的键，唯一索引会将其作为null存储，因此当插入多个不包含索引字段的文档的时候会导致插入失败。

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，创建唯一索引的时候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

创建复合唯一索引的时候，单个值可以相同，只要所有键的值组合起来不同就好。

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 }

5.4

集合名和索引名加起来不能超过127字节。

修改索引：

> db.blog.ensureIndex({title:1},{backgroud:true}) //backgroud选项可以使得创建索引的工作在后台运行，而且不阻塞期间的请求。

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

如果阻塞建立索引的话，会使得索引建立的非常快，但是也意味着在此期间的所有请求都不能应答。后台创建索引也会增加负载，好在不会让服务器停机。

为已有的文档创建索引比先创建索引在插入所有文档要稍快一点。

删除指定名字的索引：

> db.runCommand({"dropIndexes":"foo","index":"url\_1"}) //参数包括集合名和索引名

{

"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.blog.dropIndex("title\_1") //这也是删除索引的一种方式

{ "nIndexesWas" : 2, "ok" : 1 }

另一种删除索引的方式就是删除集合，删除集合的所有文档并不会删除索引。

> 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 } }...

默认的地理空间索引的值的范围在-180~180之间，可以在创建索引的时候指定：

> db.map.ensureIndex({gps:"2d"},{min:0,max:1000}) //第二个参数用来指定索引值的范围

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

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，所以最好使用limit命令

1. 也可以使用geoNear来完成相同的操作，普通查询和使用命令查询返回的形式不同，而且命令查询还会返回每个文档到查询点的距离。

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

{

"waitedMS" : NumberLong(0),

"results" : [

{

"dis" : 13.601470508735444, //为文档到查询点的距离

"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，那可能是mongodb数据库里面维护了这样的一个字段。

也可以传递参数，限定条件

> db.map.count({"gps.x":{$lt:30},"gps.y":{$gt:20}}) //查询内嵌文档的时候需要双引号

8

6.2 distinct

用来找出给定键的所有不同的值，使用时必须指定集合和键，如果为该键建立了索引的话就会非常的快，而且结果还是按照一定顺序排列的。

> 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"

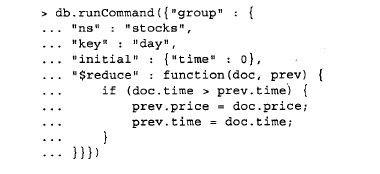
},

"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，然后可以通过聚合每一组内的文档，产生一个结果文档。



也可以这样执行命令：

db.map.group({/…/}) //里面是类似runCommand时的参数

6.3.1 使用完成器-finalizer

完成器可以把累加器文档变成任意形态，甚至变换成非文档（如数字和字符串）

finalize附带一个函数，在每组结果传递到客户端之前被调用一次，可以对结果进一步精简。

finalize能修改传递的参数也能够返回新值。

6.3.2 将函数作为键使用

定义分组函数就要用到$keyf键：

>db.posts.group(

“ns”:”posts”,

“$keyf”:function(x){return x.category.toLowerCase(); }, //这样categroy无论大小写都不被作为分组的依据

…

)

6.4 mapreduce

map(映射)—shuffle(洗牌)—reduce(化简)

使用mapreduce的代价就是速度。

要作为后台任务来运行mapreduce

mapreduce的运行会创建一个保存结果的集合，可以对这个集合进行实时的查询。

6.4.1 例1 找出集合中的所有键

map函数使用emit函数返回要处理的值。

emit会给mapreduce一个键（类似于前面group所使用的键）和一个值。这里用emit将文档某个键的计数（count）返回（{count:1}）。我们想为每个键单独计数，所以为文档的每个键调用一次emit。this是对当前映射的文档的引用。

> var map=function(){

... for(var key in this){ //this就是对当前文档的引用

... emit(key,{count:1}); //emit将文档某个键的计数（count）返回

... }};

> map

function (){

for(var key in this){

emit(key,{count:1});

}}

经过map函数的作用，就有了许许多多的{count:1}的文档，每一个都与集合中的一个键相关。

这种由一个或多个{count:1}文档组成的数组，会传递给reduce函数。

reduce函数有两个参数，一个是key，也就是emit返回的第一个值，还有另外一个数组，由一个或者多个对应于键的{count:1}文档组成。

> var reduce=function(key,emits){

... total=0;

... for(var i in emits){

... total+=emits[i].count;

... }

... return {"count":total};

... }

> reduce

function (key,emits){

total=0;

for(var i in emits){

total+=emits[i].count;

}

return {"count":total};

}

reduce一定要能被反复调用，不论是映射环节（map）还是简化环节(reduce),所以reduce返回的文档必须能作为reduce的第二个参数的一个元素。

reduce应该能处理emit文档和其他reduce结果的各种组合。

mapreduce会返回一个文档，包含很多与操作有关的信息。

> db.runCommand({

... mapreduce:"EdgeBase",

... map:map,

... reduce:reduce,

... out:"mapreduce\_edgebase\_01"

... })

{

"result" : "mapreduce\_edgebase\_01", //这是存放结果集的集合名

"timeMillis" : 1453, //这是运算总共耗时

"counts" : {

"input" : 9296, //发送到map函数的文档个数

"emit" : 55776, //在map函数中emit方法被调用的次数

"reduce" : 558, //

"output" : 6 //结果集合中包含的文档的数量

},

"ok" : 1

}

> show tables

EdgeBase

StarNode

mapreduce\_edgebase\_01 //mapreduce新生成的结果集

> db.mapreduce\_map\_01.find() //结果集中的values就是结果，mapreduce的结果集会自动带有分组的key

{ "\_id" : "\_id", "value" : 100 }

{ "\_id" : "x", "value" : 100 }

{ "\_id" : "y", "value" : 100 }

mongodb和mapreduce：

mapreduce可选的键：

finalize函数

keeptemp：布尔

out:字符串

query:文档，在发往map前先用执行条件过滤文档

sort:文档

limit：整数

scope:文档，javascript代码中要用到的变量

verbose:布尔，是否产生更加详尽的服务日志。

> db.runCommand({ mapreduce:"StarNode", map:map, reduce:reduce,limit:10000, verbose:true

,out:"out\_05" })

{

"result" : "out\_05",

"timeMillis" : 4212,

"timing" : {

"mapTime" : 2377,

"emitLoop" : 4122,

"reduceTime" : 1240,

"mode" : "mixed",

"total" : 4212

},

"counts" : {

"input" : 10000,

"emit" : 160189,

"reduce" : 1641,

"output" : 21

},

"ok" : 1

}

1. 进阶指南

7.1 数据库命令

> db.runCommand({"getLastError":1})//最近执行的情况

{

"connectionId" : 1,

"n" : 0,

"syncMillis" : 0,

"writtenTo" : null,

"err" : null,

"ok" : 1

}

7.1.1 命令的工作原理

例如删除:db.foo.drop()

实际执行的是drop函数，同样可以通过命令来执行：

> db.runCommand({"drop":"c"})

{ "ns" : "test.c", "nIndexesWas" : 2, "ok" : 1 }

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

{

"connectionId" : 1,

"n" : 0,

"syncMillis" : 0,

"writtenTo" : null,

"err" : null,

"ok" : 1

}

> db.runCommand({"drop":"c"})//删除不存在的集合会报错

{ "ok" : 0, "errmsg" : "ns not found", "code" : 26 }

Mongodb中的命令实际是作为一种特殊类型的查询执行的，这个查询是对$cmd集合来执行。

RunCommand仅仅是接受命令文档，执行等价查询，因此drop调用实际上是这样的：

> db.$cmd.findOne({"drop":"analytics"})

{ "ns" : "test.analytics", "nIndexesWas" : 1, "ok" : 1 }

> db.runCommand({"drop":"analytics"})

{ "ok" : 0, "errmsg" : "ns not found", "code" : 26 }

7.1.2 命令参考

得到命令列表：1、在shell中db.listCommands()

2、在浏览器中接口http://127.0.0.1/\_commands

常用命令:

buildInfo:返回mongodb服务器的版本号和主机的操作系统

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

{

"version" : "3.2.9",

"gitVersion" : "22ec9e93b40c85fc7cae7d56e7d6a02fd811088c",

"targetMinOS" : "Windows 7/Windows Server 2008 R2",

"m

...

CollStats:返回指定集合的统计信息

db.runCommand({“collStats”:”jfoo”})

Distinct:列出指定集合中满足查询条件的文档中指定键的所有不同值

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

{

"waitedMS" : NumberLong(0),

"values" : [

"aple",

"banana",

"cherry"

],

"stats" : {

"n" : 1,

"nscanned" : 0,

"nscannedObjects" : 5,

"timems" : 0,

"planSummary" : "COLLSCAN"

},

"ok" : 1

}

Drop:删除集合所有数据

> db.runCommand({"drop":"foo"})

{ "ns" : "test.foo", "nIndexesWas" : 1, "ok" : 1 }

DropDatabase删除当前数据库所有数据;

db.runCommand({“dropDatabase”:1})

dropIndexes:删除集合里面名称为index\_name的索引，如果名称为’\*’则删除所有索引

> db.runCommand({“dropIndexes”:”test”,”index”:”index\_name”})

findAndModify

getLastError:查看对本集合执行的最后一次操作的信息

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

{

"connectionId" : 1,

"n" : 0,

"syncMillis" : 0,

"writtenTo" : null,

"err" : null,

"ok" : 1

}

IsMaster:检查本服务器是主服务器还是从服务器

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

{

"ismaster" : true,

"maxBsonObjectSize" : 16777216,

"maxMessageSizeBytes" : 48000000,

"maxWriteBatchSize" : 1000,

"localTime" : ISODate("2016-10-17T14:22:28.497Z"),

"maxWireVersion" : 4,

"minWireVersion" : 0,

"ok" : 1

}

ListCommands：列出所有的命令

> db.runCommand({“listCommands”:1})

listDatabases:列出服务器上所有数据

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

{

"ok" : 0,

"errmsg" : "listDatabases may only be run against the admin database.",

"code" : 13

}

Ping：检查服务器连接是否正常

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

{ "ok" : 1 }

RenameCollection:将集合a重命名为b,包含数据库的名.集合名

RepairDatabase:修正及压缩当前数据库，这个操作可能非常耗时

ServerStatus:返回这台服务器的管理统计信息;

db.runCommand({“serverStatus”:1})

7.2 固定集合

Mongodb还支持另外一种集合：固定集合，要实现创建并且大小固定。固定集合很像环形队列，如果空间不足，最早的文档就会被删除，为新的文档腾出空间。

默认情况下固定集合没有索引

7.2.2 创建固定集合

固定集合必须要在使用前显示的创建：

> db.createCollection("my\_collection",{capped:true,size:10000})

{ "ok" : 1 }

也可以通过转换普通集合来创建固定大小的集合，使用命令converToCapped:

> db.runCommand({"convertToCapped":"food",size:10000})//将food集合转换为固定大小

{ "ok" : 1 }

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

{

"connectionId" : 1,

"n" : 0,

"syncMillis" : 0,

"writtenTo" : null,

"err" : null,

"ok" : 1

}

7.2.3 自然排序

固定集合的排序方式为自然排序，即文档插入时的顺序。查询时就是按照此顺序返回的，也可以指定反向返回。

> db.food.find().sort({$natural:-1})//自然顺序反向排列

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

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

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

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

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

7.2.4 尾部有标

7.3 Grid:存储文件

Mongodb分配数据文件空间时以2GB为一块

7.3.1 开始使用GridFS:mongofiles

命令行工具：mongofiles,mongodb自带

D:\mongodb\bin>mongofiles put foo.txt //添加文件

2016-10-18T07:56:25.750+0800 connected to: localhost

added file: foo.txt

D:\mongodb\bin>mongofiles list //列示文件系统中的文件

2016-10-18T07:57:13.215+0800 connected to: localhost

foo.txt 15

D:\mongodb\bin>del foo.txt //删除磁盘上的foo.txt文件

D:\mongodb\bin>mongofiles get foo.txt //将文件写入到磁盘，与put相反

2016-10-18T07:58:53.689+0800 connected to: localhost

finished writing to foo.txt

D:\mongodb\bin>more foo.txt

"hello world"

Search命令用来按照文件名来查找GridFS中的文件

D:\mongodb\bin>mongofiles search foo

2016-10-18T08:02:03.474+0800 connected to: localhost

foo.txt 15

delete命令用来从GridFS中删除一个文件

D:\mongodb\bin>mongofiles delete foo.txt

2016-10-18T08:03:39.183+0800 connected to: localhost

successfully deleted all instances of 'foo.txt' from GridFS

D:\mongodb\bin>mongofiles list

2016-10-18T08:03:54.883+0800 connected to: localhost

7.4 服务器端脚本

可以通过db.eval函数来执行javascript脚本

发送代码有两种选择：

1. 封装进一个函数

> db.eval("function(){return 1;}")

WARNING: db.eval is deprecated

1

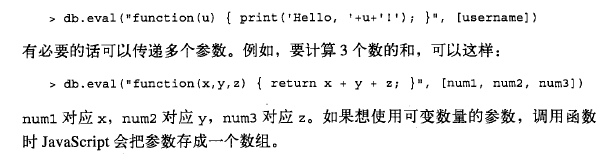
1. 不封装

> db.eval("return 1")

WARNING: db.eval is deprecated

1

要传递参数就必须使用函数封装的形式，参数作为db.eval函数的第二个参数传递：



7.4.2 存储javascript

System.js集合，存储javascript变量，可以在全局调用这些变量

> db.system.js.insert({"\_id":"x","value":1}) //存入x,值为1

WriteResult({ "nInserted" : 1 })

> db.system.js.insert({"\_id":"y","value":2})

WriteResult({ "nInserted" : 1 })

> db.system.js.insert({"\_id":"z","value":3})

WriteResult({ "nInserted" : 1 })

> db.eval("return x+y+z;") //调用变量

WARNING: db.eval is deprecated

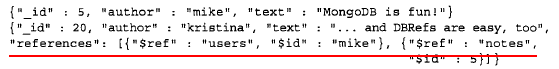
6

System.js还可以存储一些javascript代码，程序等

7.5 数据库引用

DBRef:就像url,唯一确定一个到文档的引用。他是一个内嵌文档，是内嵌在普通集合中的，作为集合的一部分

{$ref:集合名,$id:id值,$db:数据库名}，数据库项是可选项，而且顺序必须是集合名-id值



附录B MongoDB shell

启动shell时指定端口

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

附录C 深入mongodb内部

C1：BSON---Binary JSON，一种轻量二进制格式。

使用BSON格式的三个主要目标：

效率、可遍历性、性能

C2：mongo传输协议

C3：数据文件

C4：命名空间和数据域：每个数据库都是按照命名空间组织的，每个集合的文档都有自己的命名空间。

C5:内存映射存储引擎：这时mongodb默认的存储引擎，即当服务程序启动后将所有的数据文件映射到内存。