**目录**

[1. 多条件查询 1](#_Toc476312440)

[2. 虑重 1](#_Toc476312441)

[3. 统计 1](#_Toc476312442)

[4. 分组 1](#_Toc476312443)

[4.1group 1](#_Toc476312444)

[4.1.1简单分组 1](#_Toc476312445)

[4.1.2分组多记录合并 1](#_Toc476312446)

[4.2aggregate 1](#_Toc476312447)

[4.2.1$unwind 1](#_Toc476312448)

[4.2.2复杂aggregate操作1 1](#_Toc476312449)

[4.2.3负责aggregate操作2 1](#_Toc476312450)

[4.2.4总结 1](#_Toc476312451)

[4.3mapreduce操作 1](#_Toc476312452)

[4.3.1单key运算 1](#_Toc476312453)

[4.3.2复合key运算 1](#_Toc476312454)

[4.4小结 1](#_Toc476312455)

[5. 多表联合查询 1](#_Toc476312456)

[5.1普通手动联合操作 1](#_Toc476312457)

[5.2DBRef 1](#_Toc476312458)

[6. 指定数据只保留1个 1](#_Toc476312459)

[7. 参考资料 1](#_Toc476312460)

jar cvfm MyWordCount.jar manifest.data \*.class

汉字

习惯于Oracle，MySQL等关系型数据库的SQL操作，对于初次接触MongoDB的查询语法而言，不是那么容易转换，如下简单介绍常见的SQL操作与MongoDB之间的转换。

对于这些操作尤其是聚合操作，MongoDB的低版本支持比较弱，很多选项不支持，使用时务必对照官方文档进行参考。

# 多条件查询

db.student.find({"age":19, "name":/.\*ang.\*/i});

普通条件比对和正则匹配。

# 虑重

db.student.distinct(“age”);

MongoDB中貌似不能处理多列同时distinct。

# 统计

db.student.count({“age”:20});

统计年龄为20的人数。

# 分组

MongoDB提供了好几种可以进行分组的方法，以各自函数名执行均返回游标，以runCommand则返回普通信息。

如下：

## 4.1group

db.runCommand(

{group:

{

ns: <namespace>,

key: <key>,

$reduce: <reduce function>, // function(curr, result)进入的数据被分组过，因此在这里操作就相当于对同一个组的数据操作

$keyf: <key function>, // function(doc)和key的作用相同，有时需要预处理组成一些复杂的key会用到keyf

cond: <query>, // 类似于select中的where，对数据进行过滤

finalize: <finalize function> // function(result)可以在这里对产生的结果再进行处理

}

}

);

或者

db.collection.group(

{

key: <key>,

$reduce: <reduce function>, // function(curr, result)进入的数据被分组过，因此在这里操作就相当于对同一个组的数据操作

$keyf: <key function>, // function(doc)和key的作用相同，有时需要预处理组成一些复杂的key会用到keyf

cond: <query>, // 类似于select中的where，对数据进行过滤

finalize: <finalize function> // function(result)可以在这里对产生的结果再进行处理

}

);

### 4.1.1简单分组

db.runCommand(

{group:

{

ns:'student',

key:{'age':1, 'sex':1},

$reduce:function (cur, result) {

result.total += 1;

},

initial:{total:0}

}

}

);

以类似于select age, sex, count(1) from student group by age, sex;

### 4.1.2分组多记录合并

db.student.group(

{

"key":{"age":true},

"initial":{"person":""},

"$reduce":function(cur,prev) {

prev.person += cur.name;

}

}

);

类似于select中先group by然后将某列合并成一行。

## 4.2aggregate

db.runCommand(

{

aggregate: "<collection>", // 要处理的表

pipeline: [ <stage>, <...> ], // 管道过滤条件

explain: <boolean>, // 返回一些处理信息

allowDiskUse: <boolean>, // 使用内存最大大小有限制，此处允许在磁盘缓存数据

cursor: <document> // 结果集单条大于16M时，可以使用游标解决这个问题

}

);

或者

db.collection.aggregate(

{

pipeline: [ <stage>, <...> ], // 管道过滤条件

explain: <boolean>, // 返回一些处理信息

allowDiskUse: <boolean>, // 使用内存最大大小有限制，此处允许在磁盘缓存数据

cursor: <document> // 结果集单条大于16M时，可以使用游标解决这个问题

}

);

通过aggregate提供的pipeline一些列相关操作，可以实现各种复杂的sql中操作。pipeline项可选的操作有：$project、$match、$redact、$limit、$skip、$unwind、$group、$geoNear、$out等。

$project、$match、$redact、$limit、$skip对数据进行过滤；

$unwind将array拆解，没有array的将被过滤掉；

$group、$sort将数据分组（分组里面可以使用$sum、$avg、$first、$max、$min等来做运算），排序；

$our输出到文档；

### 4.2.1$unwind

{ "\_id" : 1, "item" : "ABC1", sizes: [ "S", "M", "L"] }

db.inventory.aggregate( [ { $unwind : "$sizes" } ] )

{ "\_id" : 1, "item" : "ABC1", "sizes" : "S" }

{ "\_id" : 1, "item" : "ABC1", "sizes" : "M" }

{ "\_id" : 1, "item" : "ABC1", "sizes" : "L" }

### 4.2.2复杂aggregate操作1

这个例子基本应用了aggregate的所有操作：

db.runCommand(

{

aggregate: "student",

pipeline: [

{$project: {\_id:1, age:1, sex:1, name:1}},

{$match: {age:{$gt:10}}},

{$limit:10},

{$skip:0},

{$group:

{

\_id: {new\_sex:"$sex", new\_age:"$age"},

age\_sum:{$sum:"$age"},

count:{$sum:1},

name\_list:{$push:"$name"}

}

},

{$sort:{age\_sum:1}}

]

}

);

如下是一个解释：

db.runCommand(

{

aggregate: "student",

pipeline: [

// (1)用\_id，age，sex, name形成vt1

{$project: {\_id:1, age:1, sex:1, name:1}},

// (2)根据vt1，将age大于10的选出来形成vt2

{$match: {age:{$gt:10}}},

// (3)从vt2中选出10条

{$limit:10},

// (4)从第0个开始，选出10条，形成vt3

{$skip:0},

// (5)根据vt3进行分组，计算形成vt4，vt4中有\_id、age\_sum、count、name\_list

{$group:

{

// 对vt3中分组，key是sex，age。new\_sex，new\_age是新名称

\_id: {new\_sex:"$sex", new\_age:"$age"},

// 统计同一类key所有age总和

age\_sum:{$sum:"$age"},

// 统计同一类key的数量

count:{$sum:1},

// 同一类key的name列组成数组

name\_list:{$push:"$name"}

}

},

// (6)根据vt4对age\_sum列进行增序排序，形成最终结果输出

{$sort:{age\_sum:1}}

]

}

);

### 4.2.3负责aggregate操作2

在4.2.2的基础上再次嵌套各种操作，如下：

db.runCommand(

{

aggregate: "student",

pipeline: [

{$project: {\_id:1, age:1, sex:1, name:1}},

{$match: {age:{$gt:10}}},

{$limit:10},

{$skip:0},

{$group:

{

\_id: {new\_sex:"$sex", new\_age:"$age"},

age\_sum:{$sum:"$age"},

count:{$sum:1},

name\_list:{$push:"$name"}

}

},

{$sort:{age\_sum:1}},

**{$group:**

**{**

**\_id:{new\_age\_sum:"$age\_sum"},**

**max\_count:{$max:"$count"}**

**}**

**},**

**{$sort:{max\_count:-1, \_id:1}}**

]

}

);

解释，上述命令是在4.2.2的基础上，添加了蓝色部分代码，蓝色代码对上述结果再次进行分组排序。由此也可以看出pipeline中的各种操作是按照顺序进行的，我们可以在pipeline中放多个同类命令，上述命令就有2个$group操作，2个$sort操作。

### 4.2.4总结

aggregate命令功能强大，可实现众多命令联合操作，非常剽悍。

## 4.3mapreduce操作

db.runCommand(

{

mapReduce: <collection>,

map: <function>,

reduce: <function>,

finalize: <function>,

out: <output>,

query: <document>,

sort: <document>,

limit: <number>,

scope: <document>,

jsMode: <boolean>,

verbose: <boolean>

}

);

或者

db.collection.mapReduce(

<map>,

<reduce>,

{

out: <collection>,

query: <document>,

sort: <document>,

limit: <number>,

finalize: <function>,

scope: <document>,

jsMode: <boolean>,

verbose: <boolean>

}

);

### 4.3.1单key运算

// 定义map函数

var map=function(){

// url作为单key

**emit(this.url,{"count":1});**

}

// 定义reduce函数

var reduce=function(key,values){

var total=0;

for(var i=0; i < values.length; i++){

total+=values[i].count;

}

return {count:total}

}

/ /执行mapreduce函数,其中out的值是存储执行结果的集合

db.runCommand(

{

"mapreduce":"visit",

"map":map,

"reduce":reduce,

"query":{"vtime":{"$gte":1412611200,"$lte":1413907119}},

"out":"test.tmp"

}

);

### 4.3.2复合key运算

// map函数实现

map=function (){

// user和sku作为复合key

**emit({user:this.user, sku:this.sku},{count:1})**

}

// reduce函数实现

reduce=function (key,values){

var cnt=0;

// 此处使用javascript语法遍历，function(val)中val是数据中的每一个元素

values.forEach(

function(val) {

cnt+=val.count;

}

);

return {"count":cnt};

}

db.test.mapReduce(map,reduce,{out:"mr2"})

## 4.4小结

group…by功能简单，使用方便；

aggregate可以进行比较复杂的运算，但相比mapreduce方式还是比较简单；

mapreduce功能复杂强大，但效率较慢。

# 多表联合查询

在这里就得吐槽一下了，相比SQL脚本的简单易写，MongoDB中的这些复杂操作确实麻烦。我所了解的MongoDB提供了2中方式用以表间数据查询操作。

## 5.1普通手动联合操作

只查找 1个的情况：

temp = db.class.findOne({name:"level\_2"});

for(var post = db.student.find({"class\_level":temp.name}); post.hasNext();){ printjson(post.next().name); }

整个表的某列关联（m\*n的复杂度）：

var temp=db.class.find({});

temp.forEach(function(x){

for(var post = db.student.find({"class\_level":x.name}); post.hasNext();) {

printjson(post.next().name);

}

});

这相当于把javascript语法拿来直接用了（效果同上）：

var temp=db.class.find({});

while(temp.hasNext()) {

x = temp.next();

for(var post = db.student.find({"class\_level":x.name}); post.hasNext();) {

printjson(post.next().name);

}

}

## 5.2DBRef

// 保存查出的信息

> author=db.authors.find({name:"chenzhou"})[0]

{

"\_id" : ObjectId("5030ba7621bdee44765b2147"),

"name" : "chenzhou",

"email" : "chenzhou1025@126.com"

}

// 引用刚才的信息，插入数据到其他表中

> db.posts.insert({"title":"Hello Mongodb DBRef1", authors:[new DBRef('authors',author.\_id)]})

> db.posts.insert({"title":"Hello Mongodb DBRef2", authors:[new DBRef('authors',author.\_id)]})

// 根据表2查询表1中的信息

> db.posts.find({"title":"Hello Mongodb DBRef1"})[0].authors[0].fetch()

{

"\_id" : ObjectId("5030ba7621bdee44765b2147"),

"name" : "chenzhou",

"email" : "chenzhou1025@126.com"

}

# 指定数据只保留1个

在student表中每个id只保留一行记录（这是某个网友的一个错误答案，错误在于把batchSize当做了所有数据，以至于并没有将所有数据都去重）：

var rs = db.student.aggregate([{$group : {\_id:"$id",count:{$sum : 1}}},{$match : {count : {$gt : 1}}}]);

for (var i in rs.\_batch) {

var r = rs.\_batch[i].count;

var id = rs.\_batch[i].\_id;

print(id + ":" + r);

for (var j = 0; j < r - 1; j++) {

db.student.remove({"id" : id}, true);

}

}

可以通过print查看返回的数据结构，便于我们查看，比如print(rs)。不过此行为需要在robomongo等可视化工具中才能看到。

正确的方式如下：

var rs = db.student.aggregate([{$group:{\_id:"$id", count:{$sum:1}}}, {$match:{count:{$gt:1}}}]);

while(rs.hasNext()) {

var item = rs.next();

print(item.\_id + " " + item.count);

for (var i = 0; i < item.count - 1; i++) {

db.student.remove({"id" : item.\_id}, true);

}

}

：如果数据量比较大或者要删除的重复数据项很多，还是建议用distinct把符合条件的数据抽出来放在一个临时表中，把原始表删除（这个速度很快）。然后把表名称修改和原始相同，这样速度上会快很多，是不是有点智商逆天的感觉呢，。

var rs = db.student.distinct("id");

for (var item\_id in rs) {

//print(typeof(item\_id));

var n\_id = parseInt(item\_id);

var new\_item = db.student.findOne({"id":n\_id});

db.student\_back.insert(new\_item);

}

db.student.drop();

db.student\_back.renameCollection("student");

：在js模式使用mongodb的命令时，值的判断一定要注意类型。

# 参考资料

1. http://www.cnblogs.com/Joe-T/p/4264910.html
2. https://docs.mongodb.com/v2.6/reference/command/group/
3. https://docs.mongodb.com/v2.6/reference/command/nav-aggregation/