**目录**

[1. MongoDB中MapReduce概述 1](#_Toc456526920)

[1.1示例1：统计每段年龄人数 2](#_Toc456526921)

[1.2示例2：词频统计 3](#_Toc456526922)

[1.3一些提升MapReduce性能的点 3](#_Toc456526923)

[1.3.1排序 5](#_Toc456526924)

[1.3.2纯js模式 5](#_Toc456526925)

[2. MongoDB中Aggregate聚合概述 7](#_Toc456526926)

[2.1语法选项 7](#_Toc456526927)

[2.2年龄人数统计示例 7](#_Toc456526928)

[ 数据规模 7](#_Toc456526929)

[ 计算语句 8](#_Toc456526930)

[ 计算结果 8](#_Toc456526931)

[2.3按照省份统计人数 8](#_Toc456526932)

[ 数据规模 8](#_Toc456526933)

[ 计算语句 9](#_Toc456526934)

[ 结算结果 9](#_Toc456526935)

[2.4模糊查询匹配地址中含有zixhiqu的数量 9](#_Toc456526936)

[ 数据规模 9](#_Toc456526937)

[ 计算语句 10](#_Toc456526938)

[ 结算结果 10](#_Toc456526939)

[3. 再说点什么 11](#_Toc456526940)

[4. 参考资料 11](#_Toc456526941)

**code block format**

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

PS：如下测试的所有数据，除了\_id的默认索引，其他列均没有索引。MongoDB的版本是3.0.10。

# MongoDB中MapReduce概述

MapReduce操作类与Hadoop中的类似，本质实际上是关系型数据库中的group…by…操作。下面介绍在MongoDB中如何进行MapReduce的操作。

MongoDB支持MapReduce，为其设置了语法格式如下：

db.runCommand({

mapreduce:<collection>,

map:<mapfunction>,

reduce:<reducefunction>,

[,query:<query filter object>]

[,sort:<sorts the input objects using this key.Useful for optimization,like sorting by the emit key for fewer reduces>]

[,limit:<number of objects to return from collection>]

[,out:<see output options below>]

[,keeptemp:<true|false>]

[,finalize:<finalizefunction>]

[,scope:<object where fields go into javascript global scope>]

[,verbose:true]

});

各个参数的含义依次如下：

Mapreduce：要操作的目标集合；

Map：映射函数（生成键值对序列，作为reduce函数参数）；

Reduce：统计函数；

Query：目标记录过滤；

Sort：目标记录排序；

Limit：限制目标记录数量；

Out：统计结果存放集合（不指定使用临时集合，在客户端断开后自动删除）；

Keeptemp：是否保留临时集合；

Finalize：最终处理函数（对reduce返回结果进行最终整理后存入结果集合）；

Scope：向map、reduce、finalize导入外部变量；

Verbose：显示详细的时间统计信息；

## 1.1示例1：统计每段年龄人数

map方法，其中emit的第1个参数是key，第2个参数是value。

var m = function(){

if(this.age>25){

emit(this.age, {name:this.name});

}

}

reduce方法，编写如下：

var r = function(key, values){

var count = 0;

values.forEach(function(){count+=1;});

return count;

}

使用map、reduce方法：

var res = db.runCommand({

mapreduce:"emp",

map:m,

reduce:r,

out:"emp\_result"

});

最终的结果会被放在名称为emp\_result这个collection中。

## 1.2示例2：词频统计

定义map函数

var map=function(){

    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,

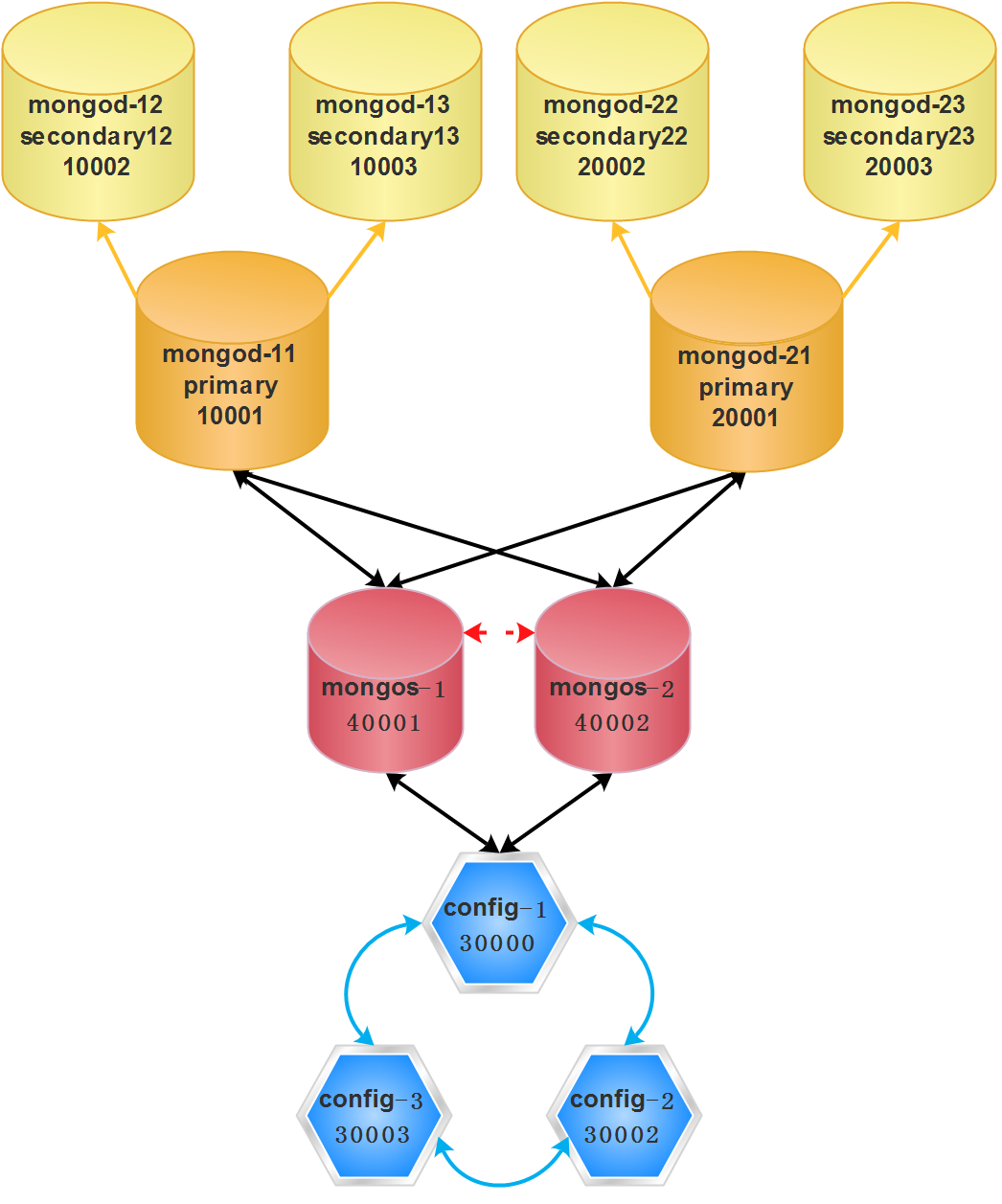
"query":{"vtime":{"$gte":1412611200,"$lte":1413907119}},

"out":"test.tmp"

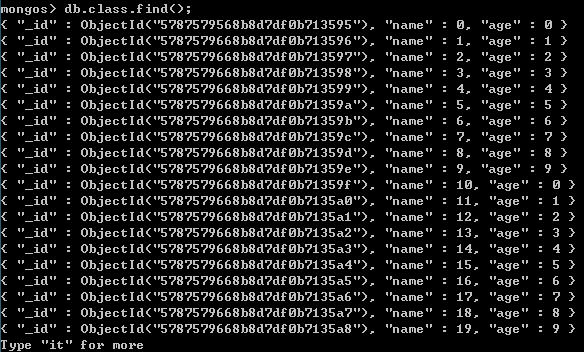
});

## 1.3一些提升MapReduce性能的点

测试环境在单台Windows7，CPUi5，内存8G，部署了一个分片集群3.0的，结构如下：



数据库表信息结构如下：



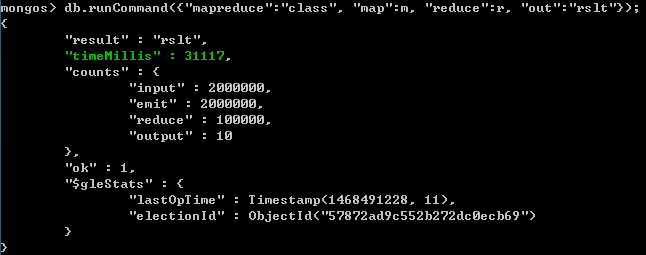
共200万数据，如下普通mapreduce的测试程序：

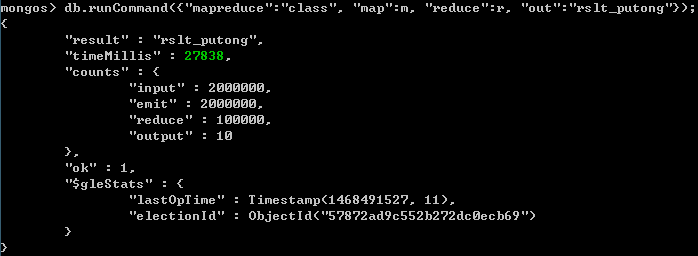
var m = function(){emit(this.age, {"count":1});}

var r = function(key, values){var total =0; for(var i=0; i<values.length; i++){total+=values[i].count;} return {count:total}}

db.runCommand({"mapreduce":"class", "map":m, "reduce":r, "out":"rslt\_putong"});

运行结果：





31s和27s，大致在30s吧。

### 1.3.1排序

var m = function(){emit(this.age, {"count":1});}

var r = function(key, values){var total =0; for(var i=0; i<values.length; i++){total+=values[i].count;} return {count:total}}

db.runCommand({"mapreduce":"class", "map":m, "reduce":r, "out":"rslt\_putong", "sort":{"age":1}});

注意排序的列必须是有索引的，作者测试时提高了性能，但是我测试反倒降低了性能。不知道为什们？

另外，鉴于其必须使用索引列，因此可以不考虑该方法来提升性能。

### 1.3.2纯js模式

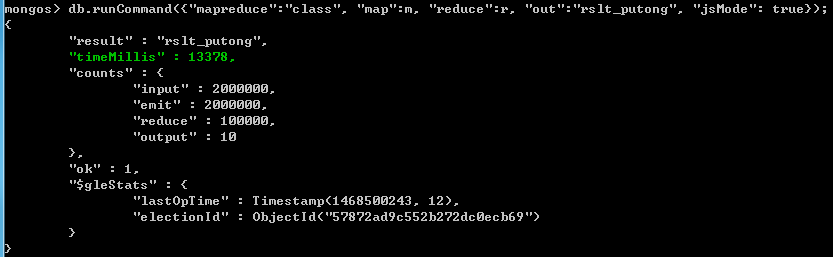
var m = function(){emit(this.age, {"count":1});}

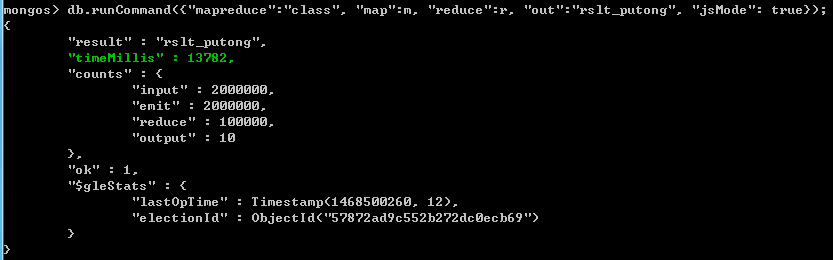
var r = function(key, values){var total =0; for(var i=0; i<values.length; i++){total+=values[i].count;} return {count:total}}

db.runCommand({"mapreduce":"class", "map":m, "reduce":r, "out":"rslt\_putong", "jsMode":true});

运行性能如下：







整体来看差不多是13s左右，相比原始方式性能提高了60%左右，幅度还是挺大的。可以通过该方式来进行性能提升。

# MongoDB中Aggregate聚合概述

聚合框架提供了一些简单的类似于sql中group…by、sum、count之类的方法，用起来比MapReduce要快，据说是C++做的。

## 2.1语法选项

db.collection.aggregate(pipeline, options)

pipeline：array类型，其中的操作符会按照从左往右的顺序执行，左边的结果是右边的输出参数，可以是如下选项：

|  |  |  |
| --- | --- | --- |
| Name | Description | 说明 |
| [$project](https://docs.mongodb.com/v3.0/reference/operator/aggregation/project/#pipe._S_project) | 预处理流入下一个操作的数据结构，可以对原有数据进行0、1处理，0表示无、1表示有。即选择某些列进行下一步操作，不必全部列都进入下一步操作。 |  |
| [$match](https://docs.mongodb.com/v3.0/reference/operator/aggregation/match/#pipe._S_match) | 就是MongoDB最原始的查询方式，使用同find方法。 |  |
| [$limit](https://docs.mongodb.com/v3.0/reference/operator/aggregation/limit/#pipe._S_limit) | 限制输出n个文档。 |  |
| [$skip](https://docs.mongodb.com/v3.0/reference/operator/aggregation/skip/#pipe._S_skip) | 与limit同用，用于输出第m-n个文档。 |  |
| [$group](https://docs.mongodb.com/v3.0/reference/operator/aggregation/group/#pipe._S_group) | 对数据进行分组操作。 | group()方法对于分组的key的数量有限制。错误信息：" group() can't handle more than 20000 unique keys "。直至我写此文当时最新版本的3.0.12也有此限制。  如果超过2W个key使用mapreduce或者aggregate来处理，但是mapreduce效率低，而aggregate可能出现单条记录超过16M的错误。 |
| [$sort](https://docs.mongodb.com/v3.0/reference/operator/aggregation/sort/#pipe._S_sort) | 对数据进行排序，-1或者1。 |  |
| [$out](https://docs.mongodb.com/v3.0/reference/operator/aggregation/out/#pipe._S_out) | 将结果写到某个文档中。 |  |

options：一些可选项，常用的就是allowDiskUses将临时结果写入磁盘文件。而explain选项的使用有限制，$group是不能使用的。

使用示例，查询并进行分组，然后排序并输出到文档：

db.employee.aggregate([{$match:{address:/zizhiqu/i}}, {$group:{\_id:"$address", count:{$sum:1}}}, {$sort:{count:-1}}, {$skip:1}, {$limit:3}, {$out:"rslt"}]);

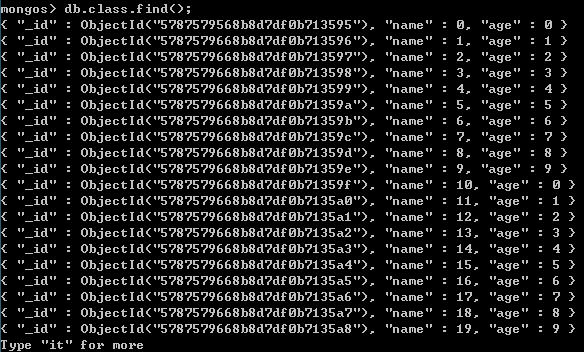
搜索address列包含zizhiqu的数据，对齐按照address列进行分组，然后降序排列，输出第[2,4]条数据，将输出写入到rslt文档中。

db.class.aggregate({$group:{\_id:"$class\_id", class\_idcount:{$sum:1}}}, {$match:{class\_idcount:2}});

## 2.2年龄人数统计示例

### 数据规模

1. 单表200w条；
2. 数据格式如下：



1. 数据非常均匀，而且年龄列每个年龄的数量都是相等的（插入时没有进行随即处理）；

### 计算语句

db.class.aggregate({$group:{\_id:"$age", age\_count:{$sum:1}}});

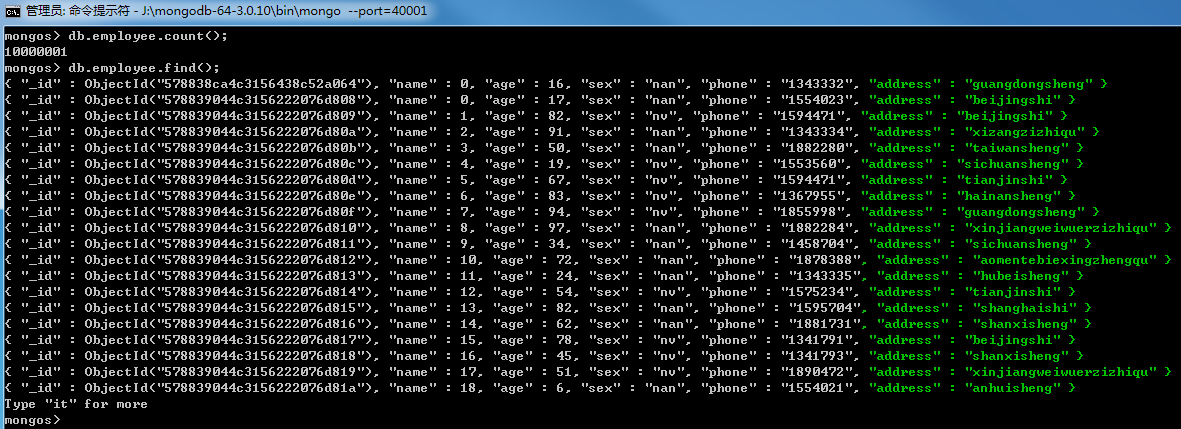
### 计算结果

时间非常快，<5s即可，具体耗时细节暂未找到统计方式。

## 2.3按照省份统计人数

### 数据规模

1. 单表1000w条；
2. 数据格式如下：



1. 每条数据比较均衡，所统计的列相比之前比较零散，每个列的数量也是参差不齐；

### 计算语句

db.employee.aggregate({$group:{\_id:"$address", age\_count:{$sum:1}}});

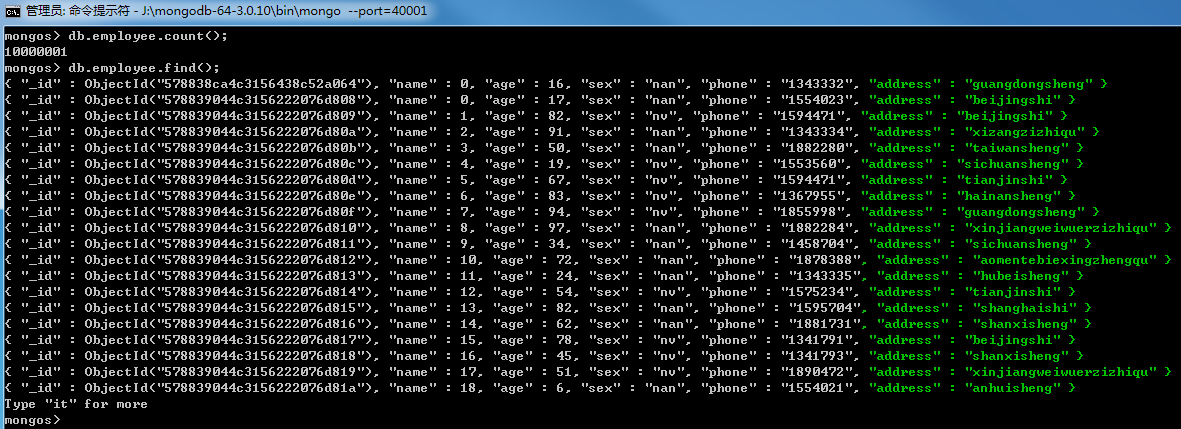
### 结算结果

耗时<60s。统计出的结果，每个分组的数量在29w-50w之间，还算是非均匀数据。

## 2.4模糊查询匹配地址中含有zixhiqu的数量

### 数据规模

1. 单表1000w条；
2. 数据格式如下：



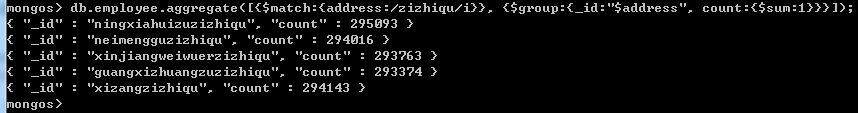
1. 每条数据比较均衡，所统计的列相比之前比较零散，每个列的数量也是参差不齐；

### 计算语句

db.employee.aggregate([{$match:{address:/zizhiqu/i}}, {$group:{\_id:"$address", count:{$sum:1}}}]);

### 结算结果

耗时<60s。统计出的结果，每个分组的数量在29w左右，基本算是非均匀数据。结果如下：



PS：

1. 之所以使用$group是因为，aggregate没有count方法。
2. 如果是单项统计，直接使用find就可以，速度也不差，对于同样的数量级find也是<60s的，如此执行：db.employee.find({address:/zizhiqu/i}).count();。

# 再说点什么

1. mapreduce

速度慢，但是可以执行较为复杂的数据逻辑处理；

1. aggregate

速度快，适合于类似于sql中group…by、count、sum之类的计算；

<2.6的版本数据量过大时，会出现“超过16M”的问题，<2.6的版本限制每个item的大小，中间结果会出现大于16M的情况，因此低版本大规模数据无法使用aggregate进行分组运算，即使3.2+的版本还是有限制，MongoDB官方给出答案，后续这个限制大小会变得放松，但是会一直存在。

# 参考资料

1. http://blog.csdn.net/yyywyr/article/details/26287483
2. http://www.iteye.com/news/28013
3. http://www.linuxidc.com/Linux/2014-11/109652.htm
4. https://docs.mongodb.com/manual/reference/method/db.collection.aggregate/
5. https://docs.mongodb.com/v3.0/reference/operator/aggregation-pipeline/