主讲老师：Fox

```
文档：2-1 MongoDB聚合操作.note
链接：http://note.youdao.com/noteshare?id=dad1b8090503bafd9cc48019bda7570f&sub=65924C3EC85544CE8BD643C74A40941E
```


1.聚合操作

1.1 单一作用聚合

1.2 聚合管道

什么是 MongoDB 聚合框架

管道（Pipeline）和阶段（Stage）

常用的管道聚合阶段

$project

$match

$count

$group

$unwind

$limit

$skip

$sort

$lookup

聚合操作案例1

聚合操作案例2

1.3 MapReduce

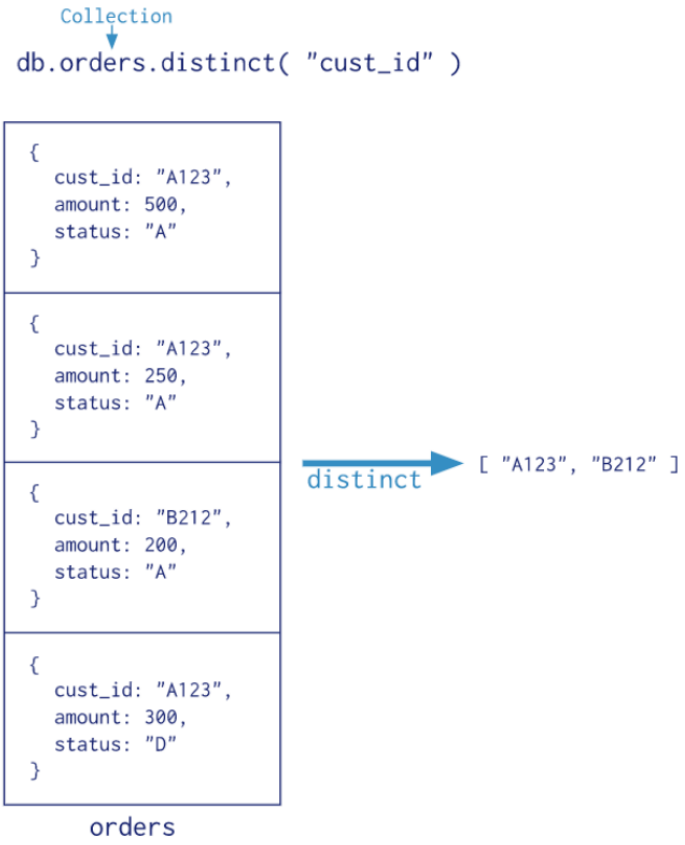
1.聚合操作

聚合操作处理数据记录并返回计算结果。聚合操作组值来自多个文档，可以对分组数据执行各种操作以返回单个结果。聚合操作包含三类：单一作用聚合、聚合管道、MapReduce。

- 单一作用聚合：提供了对常见聚合过程的简单访问，操作都从单个集合聚合文档。
- 聚合管道是一个数据聚合的框架，模型基于数据处理流水线的概念。文档进入多级管道，将 文档转换为聚合结果。
- MapReduce操作具有两个阶段：处理每个文档并向每个输入文档发射一个或多个对象的map阶段，以及reduce组合map操作的输出阶段。

1.1 单一作用聚合

MongoDB提供 db.collection.estimatedDocumentCount(), db.collection.count(), db.collection.distinct() 这类单一作用的聚合函数。 所有这些操作都聚合来自单个集合的文档。虽然这些操作提供了对公共聚合过程的简单访问，但它们缺乏聚合管道和map-Reduce的灵活性和功能。



db.collection.estimatedDocumentCount()	返回集合或视图中所有文档的计数
db.collection.count()	返回与find()集合或视图的查询匹配的文档计数 。等同于db.collection.find(query).count()构造

db.collection.distinct()	在单个集合或视图中查找指定字段的不同值，并在数组中返回结果。

```
#检索books集合中所有文档的计数
db.books.estimatedDocumentCount()
#计算与查询匹配的所有文档
db.books.count({favCount:{$gt:50}})
#返回不同type的数组
db.books.distinct("type")
#返回收藏数大于90的文档不同type的数组
db.books.distinct("type",{favCount:{$gt:90}})
```

注意：在分片群集上，如果存在孤立文档或正在进行块迁移，则db.collection.count()没有查询谓词可能导致计数不准确。要避免这些情况，请在分片群集上使用db.collection.aggregate()方法。

1.2 聚合管道

什么是 MongoDB 聚合框架

MongoDB 聚合框架（Aggregation Framework）是一个计算框架，它可以：

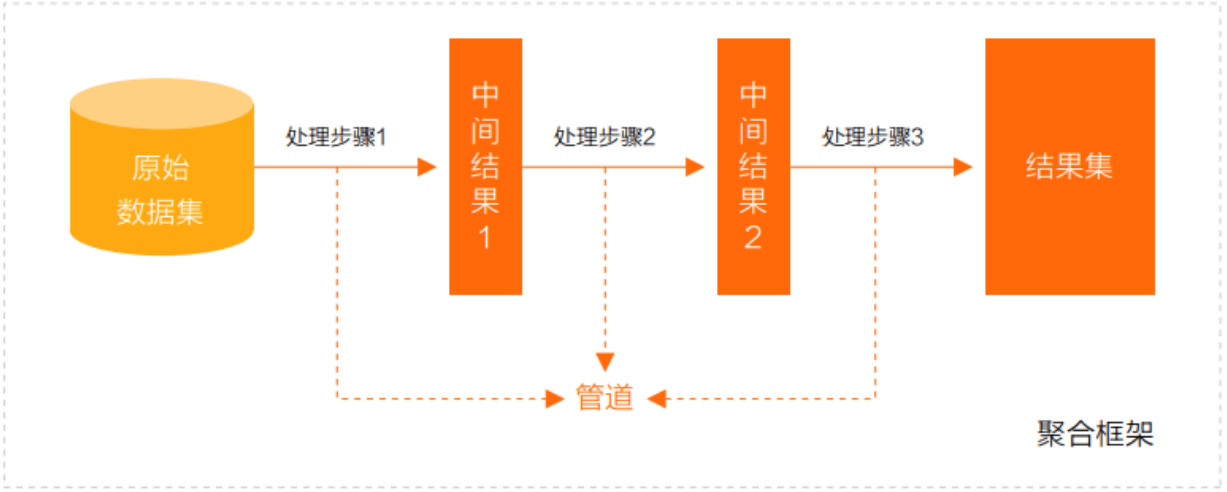
- 作用在一个或几个集合上；
- 对集合中的数据进行的一系列运算；
- 将这些数据转化为期望的形式；

从效果而言，聚合框架相当于 SQL 查询中的GROUP BY、 LEFT OUTER JOIN 、 AS等。

管道（Pipeline）和阶段（Stage）

整个聚合运算过程称为管道（Pipeline），它是由多个阶段（Stage）组成的， 每个管道：

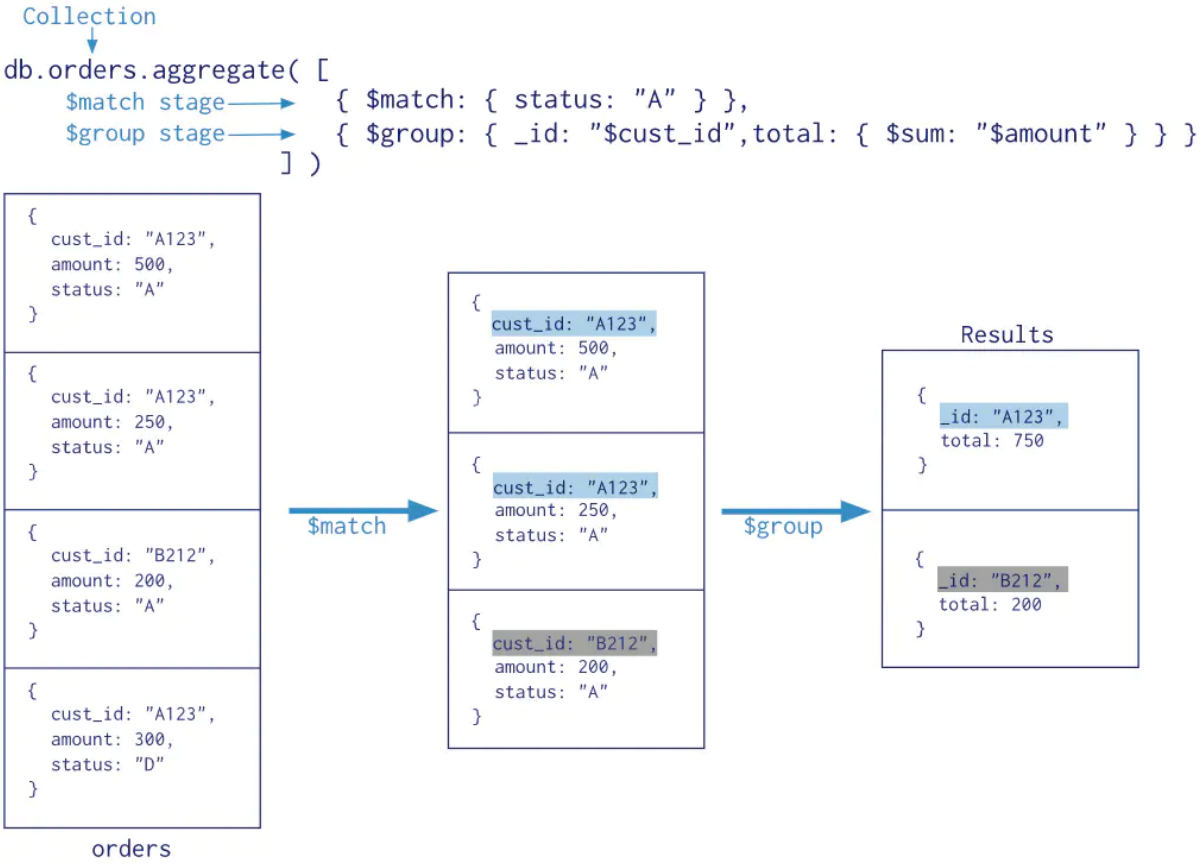
- 接受一系列文档（原始数据）；
- 每个阶段对这些文档进行一系列运算；
- 结果文档输出给下一个阶段；



聚合管道操作语法

```
pipeline = [$stage1, $stage2, ...$stageN];
db.collection.aggregate(pipeline, {options})
```

- pipelines 一组数据聚合阶段。除$out、$Merge和$geonear阶段之外，每个阶段都可以在管道中出现多次。
- options 可选，聚合操作的其他参数。包含：查询计划、是否使用临时文件、游标、最大操作时间、读写策略、强制索引等等



常用的管道聚合阶段

聚合管道包含非常丰富的聚合阶段，下面是最常用的聚合阶段

阶段	描述	SQL等价运算符
$match	筛选条件	WHERE
$project	投影	AS
$lookup	左外连接	LEFT OUTER JOIN
$sort	排序	ORDER BY
$group	分组	GROUP BY
$skip/$limit	分页	
$unwind	展开数组	
$graphLookup	图搜索	
$facet/$bucket	分面搜索	

文档： Aggregation Pipeline Stages — MongoDB Manual

聚合表达式

获取字段信息

```
$<field> ： 用 $ 指示字段路径
$<field>.<sub field> ： 使用 $ 和 . 来指示内嵌文档的路径
```

常量表达式

```
$literal :<value> ： 指示常量 <value>
```

系统变量表达式

```
$$<variable> 使用 $$ 指示系统变量
$$CURRENT 指示管道中当前操作的文档
```

数据准备

准备数据集，执行脚本

```
var tags = ["nosql","mongodb","document","developer","popular"];
var types = ["technology","sociality","travel","novel","literature"];
var books=[];
for(var i=0;i<50;i++){
 var typeIdx = Math.floor(Math.random()*types.length);
 var tagIdx = Math.floor(Math.random()*tags.length);
 var tagIdx2 = Math.floor(Math.random()*tags.length);
 var favCount = Math.floor(Math.random()*100);
```

```
 var username = "xx00"+Math.floor(Math.random()*10);
 var age = 20 + Math.floor(Math.random()*15);
 var book = {
 title: "book-"+i,
 type: types[typeIdx],
 tag: [tags[tagIdx],tags[tagIdx2]],
 favCount: favCount,
 author: {name:username,age:age}
 };
 books.push(book)
}
db.books.insertMany(books);
```

$project

投影操作， 将原始字段投影成指定名称， 如将集合中的 title 投影成 name

```
db.books.aggregate([{$project:{name:"$title"}}])
```

$project 可以灵活控制输出文档的格式，也可以剔除不需要的字段

```
db.books.aggregate([{$project:{name:"$title",_id:0,type:1,author:1}}])
```

从嵌套文档中排除字段

```
db.books.aggregate([
 {$project:{name:"$title",_id:0,type:1,"author.name":1}}
])
或者
db.books.aggregate([
 {$project:{name:"$title",_id:0,type:1,author:{name:1}}}
])
```

$match

$match用于对文档进行筛选，之后可以在得到的文档子集上做聚合，$match可以使用除了地理空间之外的所有常规查询操作符，在实际应用中尽可能将$match放在管道的前面位置。这样有两个好处：一是可以快速将不需要的文档过滤掉，以减少管道的工作量；二是如果再投射和分组之前执行$match，查询可以使用索引。

```
db.books.aggregate([{$match:{type:"technology"}}])
```

筛选管道操作和其他管道操作配合时候时，尽量放到开始阶段，这样可以减少后续管道操作符要操作的文档数，提升效率

```
db.books.aggregate([
 {$match:{type:"technology"}},
 {$project:{name:"$title",_id:0,type:1,author:{name:1}}}
```

```
4 ])
```

$count

计数并返回与查询匹配的结果数

```
1 db.books.aggregate([
2  {$match:{type:"technology"}},
3  {$count: "type_count"}
4 ])
```

$match阶段筛选出type匹配technology的文档，并传到下一阶段；

$count阶段返回聚合管道中剩余文档的计数，并将该值分配给type_count

$group

按指定的表达式对文档进行分组，并将每个不同分组的文档输出到下一个阶段。输出文档包含一个_id字段，该字段按键包含不同的组。

输出文档还可以包含计算字段，该字段保存由$group的_id字段分组的一些accumulator表达式的值。 $group不会输出具体的文档而只是统计信息。

```
1 { $group: { _id: <expression>, <field1>: { <accumulator1> : <expression1>
}, ... } }
```

- _id字段是必填的;但是，可以指定_id值为null来为整个输入文档计算累计值。
- 剩余的计算字段是可选的，并使用<accumulator>运算符进行计算。
- _id和<accumulator>表达式可以接受任何有效的表达式。

accumulator操作符

名称	描述	类比sql
$avg	计算均值	avg
$first	返回每组第一个文档，如果有排序，按照排序，如果没有按照默认的存储的顺序的第一个文档。	limit 0,1
$last	返回每组最后一个文档，如果有排序，按照排序，如果没有按照默认的存储的顺序的最后个文档。	-
$max	根据分组，获取集合中所有文档对应值得最大值。	max
$min	根据分组，获取集合中所有文档对应值得最小值。	min
$push	将指定的表达式的值添加到一个数组中。	-
$addToSet	将表达式的值添加到一个集合中（无重复值，无序）。	-
$sum	计算总和	sum
$stdDevPop	返回输入值的总体标准偏差（population standard deviation）	-

$stdDevSamp	返回输入值的样本标准偏差（the sample standard deviation）	-

$group阶段的内存限制为100M。默认情况下，如果stage超过此限制，$group将产生错误。但是，要允许处理大型数据集，请将allowDiskUse选项设置为true以启用$group操作以写入临时文件。

book的数量，收藏总数和平均值

```
db.books.aggregate([
 {$group:{_id:null,count:{$sum:1},pop:{$sum:"$favCount"},avg:{$avg:"$favCount"}}}
])
```

统计每个作者的book收藏总数

```
db.books.aggregate([
 {$group:{_id:"$author.name",pop:{$sum:"$favCount"}}}
])
```

统计每个作者的每本book的收藏数

```
db.books.aggregate([
 {$group:{_id:{name:"$author.name",title:"$title"},pop:{$sum:"$favCount"}}}
])
```

每个作者的book的type合集

```
db.books.aggregate([
 {$group:{_id:"$author.name",types:{$addToSet:"$type"}}}
])
```

$unwind

可以将数组拆分为单独的文档

v3.2+支持如下语法：

```
{
 $unwind:
 {
 #要指定字段路径，在字段名称前加上$符并用引号括起来。
 path: <field path>,
 #可选,一个新字段的名称用于存放元素的数组索引。该名称不能以$开头。
 includeArrayIndex: <string>,
 #可选，default :false，若为true,如果路径为空，缺少或为空数组，则$unwind输出文档
 preserveNullAndEmptyArrays: <boolean>
  } }
```

姓名为xx006的作者的book的tag数组拆分为多个文档

```
db.books.aggregate([
 {$match:{"author.name":"xx006"}},
 {$unwind:"$tag"}
])

db.books.aggregate([
 {$match:{"author.name":"xx006"}}
])
```

每个作者的book的tag合集

```

db.books.aggregate([
 {$unwind:"$tag"},
 {$group:{_id:"$author.name",types:{$addToSet:"$tag"}}}
])

```

案例

示例数据

```
db.books.insert([
{
 "title" : "book-51",
 "type" : "technology",
 "favCount" : 11,
 "tag":[],
 "author" : {
 "name" : "fox",
 "age" : 28
 }
},{
 "title" : "book-52",
 "type" : "technology",
 "favCount" : 15,
 "author" : {
 "name" : "fox",
 "age" : 28
 }
},{
 "title" : "book-53",
 "type" : "technology",
 "tag" : [
```

```
  "nosql",
  "document"
  ],
  "favCount" : 20,
  "author" : {
  "name" : "fox",
  "age" : 28
  }
 }])
```

测试

```
# 使用includeArrayIndex选项来输出数组元素的数组索引
db.books.aggregate([
 {$match:{"author.name":"fox"}},
 {$unwind:{path:"$tag", includeArrayIndex: "arrayIndex"}}
])
# 使用preserveNullAndEmptyArrays选项在输出中包含缺少size字段，null或空数组的文档
db.books.aggregate([
 {$match:{"author.name":"fox"}},
 {$unwind:{path:"$tag", preserveNullAndEmptyArrays: true}}
])
```

$limit

限制传递到管道中下一阶段的文档数

```

db.books.aggregate([
 {$limit : 5 }
])
```

此操作仅返回管道传递给它的前5个文档。 $limit对其传递的文档内容没有影响。

注意：当$sort在管道中的$limit之前立即出现时，$sort操作只会在过程中维持前n个结果，其中n是指定的限制，而MongoDB只需要将n个项存储在内存中。

$skip

跳过进入stage的指定数量的文档，并将其余文档传递到管道中的下一个阶段

```
db.books.aggregate([
 {$skip : 5 }
])
```

此操作将跳过管道传递给它的前5个文档。 $skip对沿着管道传递的文档的内容没有影响。

$sort

对所有输入文档进行排序，并按排序顺序将它们返回到管道。

语法：

```
{ $sort: { <field1>: <sort order>, <field2>: <sort order> ... } }
```

要对字段进行排序，请将排序顺序设置为1或-1，以分别指定升序或降序排序，如下例所示：

```
db.books.aggregate([
 {$sort : {favCount:-1,title:1}}
])
```

$lookup

Mongodb 3.2版本新增，主要用来实现多表关联查询， 相当关系型数据库中多表关联查询。每个输入待处理的文档，经过$lookup 阶段的处理，输出的新文档中会包含一个新生成的数组（可根据需要命名新key ）。数组列存放的数据是来自被Join集合的适配文档，如果没有，集合为空（即 为[ ]）

语法：

```
db.collection.aggregate([{
      $lookup: {
            from: "<collection to join>",
            localField: "<field from the input documents>",
            foreignField: "<field from the documents of the from collection>",
            as: "<output array field>"
          }
  })
```

from	同一个数据库下等待被Join的集合。
localField	源集合中的match值，如果输入的集合中，某文档没有 localField 这个Key（Field），在处理的过程中，会默认为此文档含有 localField：null的键值对。
foreignField	待Join的集合的match值，如果待Join的集合中，文档没有foreignField值，在处理的过程中，会默认为此文档含有 foreignField：null的键值对。
as	为输出文档的新增值命名。如果输入的集合中已存在该值，则会覆盖掉

注意：null = null 此为真

其语法功能类似于下面的伪SQL语句：

```
SELECT *, <output array field>
FROM collection
```

```
WHERE <output array field> IN (SELECT *
 FROM <collection to join>
 WHERE <foreignField>= <collection.localField>);
```

案例

数据准备

```
db.customer.insert({customerCode:1,name:"customer1",phone:"13112345678",address:"test1"})
db.customer.insert({customerCode:2,name:"customer2",phone:"13112345679",address:"test2"})

db.order.insert({orderId:1,orderCode:"order001",customerCode:1,price:200}
db.order.insert({orderId:2,orderCode:"order002",customerCode:2,price:400}

db.orderItem.insert({itemId:1,productName:"apples",qutity:2,orderId:1})
db.orderItem.insert({itemId:2,productName:"oranges",qutity:2,orderId:1})
db.orderItem.insert({itemId:3,productName:"mangoes",qutity:2,orderId:1})
 db.orderItem.insert({itemId:4,productName:"apples",qutity:2,orderId:2})
 db.orderItem.insert({itemId:5,productName:"oranges",qutity:2,orderId:2})
 db.orderItem.insert({itemId:6,productName:"mangoes",qutity:2,orderId:2})
```

关联查询

```
db.customer.aggregate([
 {$lookup: {
 from: "order",
 localField: "customerId",
 foreignField: "customerId",
 as: "customerOrder"
 }
 }
])

 db.order.aggregate([
  {$lookup: {
  from: "customer",
  localField: "customerCode",
  foreignField: "customerCode",
  as: "curstomer"
  }

```

```
 },
 {$lookup: {
 from: "orderItem",
 localField: "orderId",
 foreignField: "orderId",
 as: "orderItem"
 }
 }
])
```

聚合操作案例1

统计每个分类的book文档数量

```
db.books.aggregate([
 {$group:{_id:"$type",total:{$sum:1}}},
 {$sort:{total:-1}}
])
```

标签的热度排行，标签的热度则按其关联book文档的收藏数（favCount）来计算

```
db.books.aggregate([
 {$match:{favCount:{$gt:0}}},
 {$unwind:"$tag"},
 {$group:{_id:"$tag",total:{$sum:"$favCount"}}},
 {$sort:{total:-1}}
])
```

1. $match阶段：用于过滤favCount=0的文档。
2. $unwind阶段：用于将标签数组进行展开，这样一个包含3个标签的文档会被拆解为3个条目。
3. $group阶段：对拆解后的文档进行分组计算，$sum："$favCount"表示按favCount字段进行累加。
4. $sort阶段：接收分组计算的输出，按total得分进行排序。

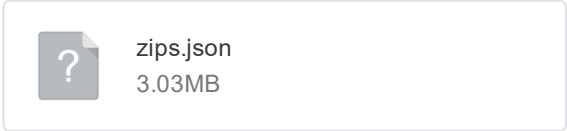
统计book文档收藏数[0,10),[10,60),[60,80),[80,100),[100,+∞)

```
db.books.aggregate([{
 $bucket:{
 groupBy:"$favCount",
 boundaries:[0,10,60,80,100],
 default:"other",
```

```
  output:{"count":{$sum:1}}
  }
}])
```

聚合操作案例2

导入邮政编码数据集:https://media.mongodb.org/zips.json



使用mongoimport工具导入数据

```
mongoimport -h 192.168.65.174 -d test -u fox -p fox --authenticationDatab
ase=admin -c zips --file D:\ProgramData\mongodb\import\zips.json
```

> h,--host：代表远程连接的数据库地址，默认连接本地Mongo数据库；
> --port：代表远程连接的数据库的端口，默认连接的远程端口27017；
> -u,--username：代表连接远程数据库的账号，如果设置数据库的认证，需要指定用户账号；
> -p,--password：代表连接数据库的账号对应的密码；
> -d,--db：代表连接的数据库；
> -c,--collection：代表连接数据库中的集合；
> -f, --fields：代表导入集合中的字段；
> --type：代表导入的文件类型，包括csv和json,tsv文件，默认json格式；
> --file：导入的文件名称
> --headerline：导入csv文件时，指明第一行是列名，不需要导入；

```
C:\WINDOWS\system32>mongoimport -h 192.168.65.174 -d test -u fox -p fox --authenticationDatabase=admin -c zips --file D:\ProgramData\mongodb\import\zips.json
2021-12-29T15:43:11.315+0800    connected to: 192.168.65.174
2021-12-29T15:43:11.845+0800    imported 29353 documents
```

返回人口超过1000万的州

```
db.zips.aggregate( [
  { $group: { _id: "$state", totalPop: { $sum: "$pop" } } },
  { $match: { totalPop: { $gte: 10*1000*1000 } } }
] )
```

这个聚合操作的等价SQL是：

```
SELECT state, SUM(pop) AS totalPop
FROM zips
GROUP BY state
HAVING totalPop >= (10*1000*1000)
```

返回各州平均城市人口

```
db.zips.aggregate( [
  { $group: { _id: { state: "$state", city: "$city" }, cityPop: { $sum:
"$pop" } } },
  { $group: { _id: "$_id.state", avgCityPop: { $avg: "$cityPop" } } }
] )
```

```
5
6 db.zips.aggregate( [
7   { $group: { _id: { state: "$state", city: "$city" }, cityPop: { $sum:
"$pop" } } }
8
9 ] )
```

按州返回最大和最小的城市

```
1 db.zips.aggregate( [
2  { $group:
3  {
4  _id: { state: "$state", city: "$city" },
5  pop: { $sum: "$pop" }
6  }
7  },
8  { $sort: { pop: 1 } },
9  { $group:
10  {
11  _id : "$_id.state",
12  biggestCity: { $last: "$_id.city" },
13  biggestPop: { $last: "$pop" },
14  smallestCity: { $first: "$_id.city" },
15  smallestPop: { $first: "$pop" }
16  }
17  },
18  { $project:
19  { _id: 0,
20  state: "$_id",
21  biggestCity: { name: "$biggestCity", pop: "$biggestPop" },
22  smallestCity: { name: "$smallestCity", pop: "$smallestPop" }
23  }
24  }
25 ] )
```

1.3 MapReduce

MapReduce操作将大量的数据处理工作拆分成多个线程并行处理，然后将结果合并在一起。MongoDB提供的Map-Reduce非常灵活，对于大规模数据分析也相当实用。

MapReduce具有两个阶段：

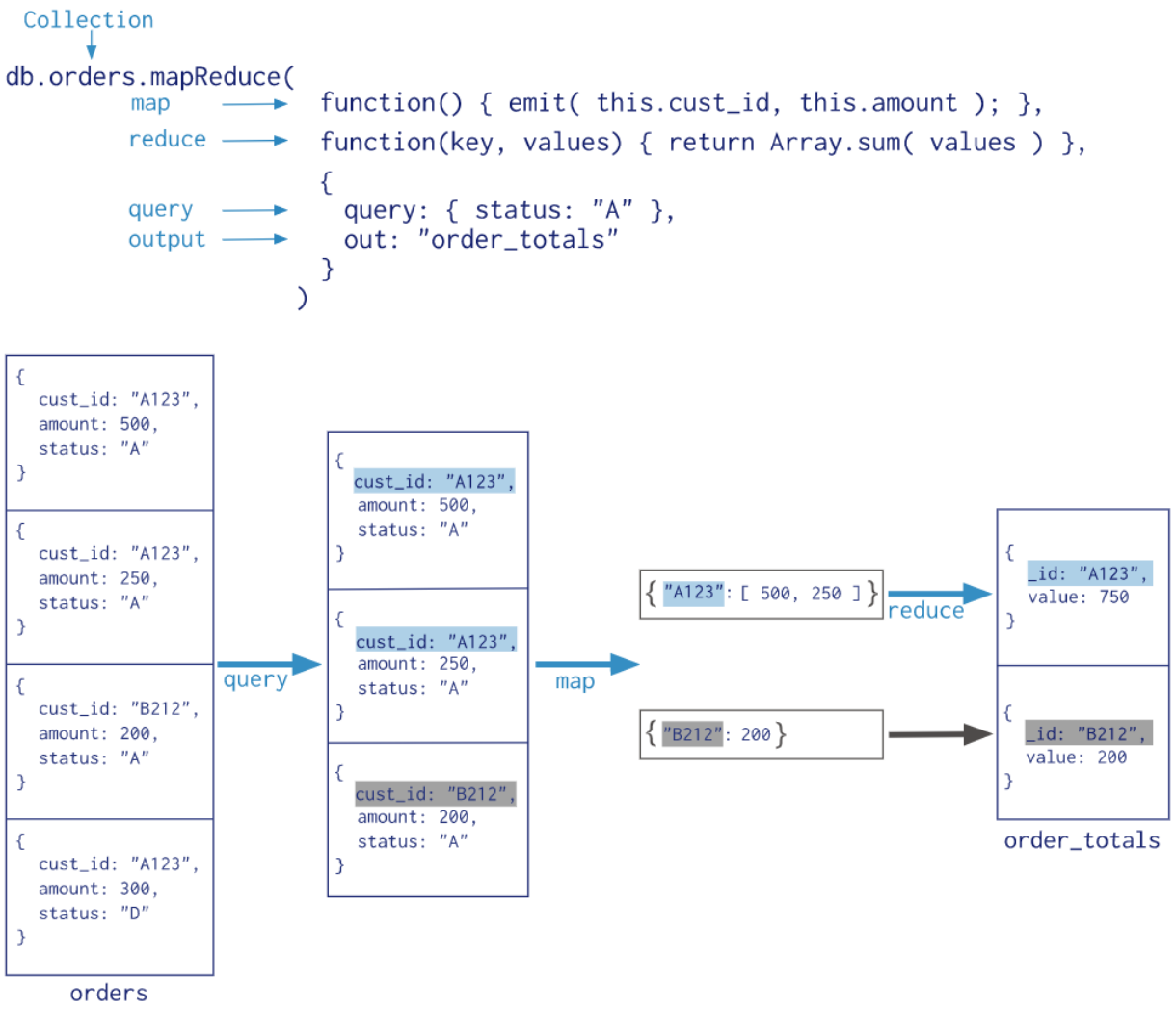
1. 将具有相同Key的文档数据整合在一起的map阶段

2. 组合map操作的结果进行统计输出的reduce阶段

MapReduce的基本语法

```
db.collection.mapReduce(
 function() {emit(key,value);}, //map 函数
 function(key,values) {return reduceFunction}, //reduce 函数
 {
 out: <collection>,
 query: <document>,
 sort: <document>,
 limit: <number>,
  finalize: <function>,
  scope: <document>,
  jsMode: <boolean>,
  verbose: <boolean>,
  bypassDocumentValidation: <boolean>
 }
)
```

- map，将数据拆分成键值对，交给reduce函数
- reduce，根据键将值做统计运算
- out，可选，将结果汇入指定表
- quey，可选筛选数据的条件，筛选的数据送入map
- sort，排序完后，送入map
- limit，限制送入map的文档数
- finalize，可选，修改reduce的结果后进行输出
- scope，可选，指定map、reduce、finalize的全局变量
- jsMode，可选，默认false。在mapreduce过程中是否将数 据转换成bson格式。
- verbose，可选，是否在结果中显示时间，默认false
- bypassDocmentValidation，可选，是否略过数据校验



统计type为travel的不同作者的book文档收藏数

```
db.books.mapReduce(
 function(){emit(this.type,this.favCount)},
 function(key,values){return Array.sum(values)},
 {
 query:{type:"travel"},
 out: "books_favCount"
 }
 )
```

```
> db.books.mapReduce(
...     function(){emit(this.tag,this.favCount)},
...     function(key,values){return Array.sum(values)},
...     {
...         query:{type:"travel"},
...         out: "books_favCount"
...     }
...  )
{ "result" : "books_favCount", "ok" : 1 }
> db.books_favCount.find()
{ "_id" : "developer", "value" : 42 }
{ "_id" : "document", "value" : 235 }
{ "_id" : "popular", "value" : 17 }
{ "_id" : "nosql", "value" : 69 }
```

从MongoDB 5.0开始，map-reduce操作已被弃用。聚合管道比映射-reduce操作提供更好的性能和可用性。Map-reduce操作可以使用聚合管道操作符重写，例如$group、$merge等。