MongoDB 聚合管道

李焕贞

河北师范大学软件学院

https://docs.mongodb.com/manual/aggregation/

本章大纲

- **聚合管道的概述**
- 聚合管道操作符
- 聚合管道的使用

聚合管道的概述

聚合操作主要用于**批量数据**处理，往往将记录按条件进行分组，然后再每组上分别进行一系列操作，例如，求最大最小值，平均值、求和等。

关系型数据库	MongoDB
SELECT cust_id, SUM(price) AS total FROM orders GROUP BY cust_id ORDER BY total	db.orders.aggregate([{ $group: { _id: "$cust_id", total: { $sum: "$price" } } }, { $sort: { total: 1 } }])

聚合管道的概述

聚合操作能够对记录进行复杂处理，主要用于数理统计和数据挖掘。在MongoDB中，聚合操作的输入时集合中的文档，输出可以是一条或多条文档。

MongoDB提供了强大的聚合功能，针对聚合功能提供了三种方式：

- 聚合管道（Aggregation PipeLine）
- 单目的聚合操作（Single Purpose Aggregation Operation）
- MapReduce编程模型

聚合管道的概述

聚合管道是MongoDB2.2版本引入的新功能，是一个全新的数据聚合框架（Aggregation Framework）。聚合管道的概念个工作方式类似于Linux中的管道命令操作符。

例如：cat number.txt | awk –F ‘,’ ‘{print $2}’ | sort –n >result.txt

聚合管道的概述

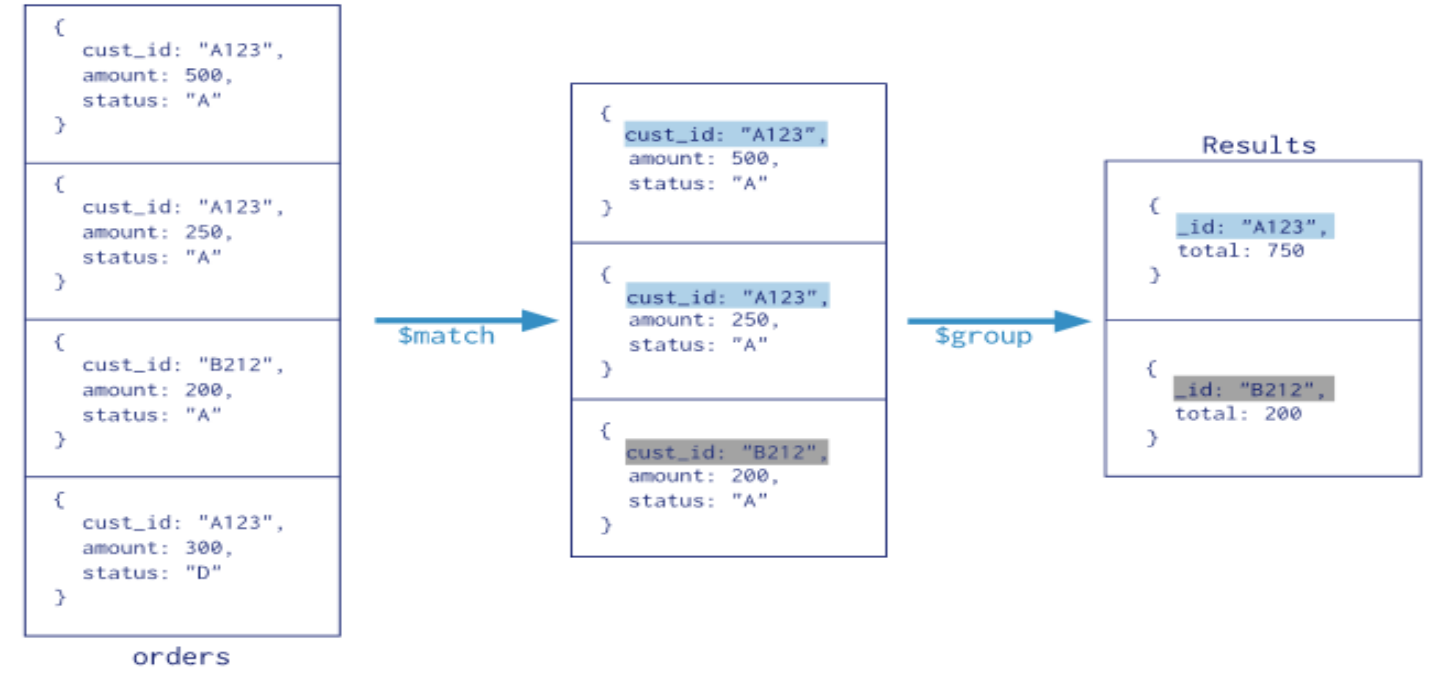
聚合管道由阶段组成，文档在一个阶段处理完毕后，聚合管道会将处理结果传递给下一个阶段。

每个阶段由阶段操作符来对文档进行相应的处理，待处理的文档会流经各个阶段，最终完成计算。计算的结果可以直接输出也可以存储到集合中。

MongoDB Shell使用db.collection.aggregate([{<stage>},…])来构建和使用聚合管道。

聚合管道的概述

```
Collection
    ↓
db.orders.aggregate( [
     $match stage ──→   { $match: { status: "A" } },
     $group stage ──→   { $group: { _id: "$cust_id",total: { $sum: "$amount" } } }
                    ] )
```



聚合管道的概述

需要注意的几个问题：

- 在每个阶段对每条输入的文档不一定都有相应的输出
- 聚合管道中，阶段是可以重复的（$out和$geoNear除外）
- 聚合管道可以在分片集合上使用
- 聚合管道函数aggregate只能作用于一个集合

本章大纲

- 聚合管道的概述
- 聚合管道操作符
- 聚合管道的使用

聚合管道操作符

聚合管道的基本功能：

- 对文档进行过滤筛选符合条件的文档
- 对文档进行变换，改变文档输出形式

每个阶段的功能使用阶段操作符定义，在每个阶段操作符中可以使用**表达式操作符**计算平均值和拼接字符串等相关操作。

聚合管道操作符-$project

{ $project: { <specification(s)> } }

仅把文档中定义好的字段作为内容输出到管道中去。

例如：

```
db.books.aggregate( [ {
 $project : {  _id:0,
address.city:1,
totalViews:{$add:[“$pageView”,10]},
newName: “$name”} } ] )
```

_id默认是显示的，不显示的指定为0或者false，否则一律显示

聚合管道操作符-$project

指定显示的字段

```
> db.books.aggregate([{$project:{title:1}}])
{ "_id" : 8751, "title" : "The Banquet" }
{ "_id" : 8752, "title" : "Divine Comedy" }
{ "_id" : 8645, "title" : "Eclogues" }
{ "_id" : 7000, "title" : "The Odyssey" }
{ "_id" : 7020, "title" : "Iliad" }
```

```
> db.books.aggregate([{$project:{title:1,_id:false}}])
{ "title" : "The Banquet" }
{ "title" : "Divine Comedy" }
{ "title" : "Eclogues" }
{ "title" : "The Odyssey" }
{ "title" : "Iliad" }
>
```

聚合管道操作符-$project

代替键名

```
> db.books.aggregate([{$project:{new_title:"$title",_id:false}}])
{ "new_title" : "The Banquet" }
{ "new_title" : "Divine Comedy" }
{ "new_title" : "Eclogues" }
{ "new_title" : "The Odyssey" }
{ "new_title" : "Iliad" }
```

```
> db.books.aggregate([{$project:{new_title:"$$CURRENT.title",_id:false}}])
{ "new_title" : "The Banquet" }
{ "new_title" : "Divine Comedy" }
{ "new_title" : "Eclogues" }
{ "new_title" : "The Odyssey" }
{ "new_title" : "Iliad" }
```

聚合管道操作符-$project

两个字段组成数组

```
> db.books.aggregate([{$project:{arr:["$title" ,"author" ]}}])
{ "_id" : 8751, "arr" : [ "The Banquet", "author" ] }
{ "_id" : 8752, "arr" : [ "Divine Comedy", "author" ] }
{ "_id" : 8645, "arr" : [ "Eclogues", "author" ] }
{ "_id" : 7000, "arr" : [ "The Odyssey", "author" ] }
{ "_id" : 7020, "arr" : [ "Iliad", "author" ] }
```

合并字段（连接字段）

```
> db.books.aggregate([{$project:{new_fild:{$concat:["$author","-","$title"]}}}])

{ "_id" : 8751, "new_fild" : "Dante-The Banquet" }
{ "_id" : 8752, "new_fild" : "Dante-Divine Comedy" }
{ "_id" : 8645, "new_fild" : "Dante-Eclogues" }
{ "_id" : 7000, "new_fild" : "Homer-The Odyssey" }
{ "_id" : 7020, "new_fild" : "Homer-Iliad" }
```

聚合管道操作符-$project

对字段进行分解

```
> db.test01.find().pretty()
{
        "_id" : 1,
        "title" : "abc123",
        "isbn" : "0001122223334",
        "author" : {
                "first" : "aaa",
                "last" : "zzz"
        },
        "copies" : 5
}
```

```
> db.test01.aggregate([{$project:{title:1,isbn:{
...  prefix:{$substr:["$isbn",0,3]},
...  group:{$substr:["$isbn",3,2]},
...  publisher:{$substr:["$isbn",5,4]},
...  title:{$substr:["$isbn",9,3]}},
...  lastName:"$author.last"}}])
{ "_id" : 1, "title" : "abc123", "isbn" : { "prefix" : "000", "group" : "11", "p
ublisher" : "2222", "title" : "333" }, "lastName" : "zzz" }
```

聚合管道操作符-$project

$substr

Deprecated since version 3.4: $substr is now an alias for $substrBytes.

{ $substrBytes: [<string expression>, <byte index>, <byte count>] }

```
> db.books.aggregate([{$project:{new_title:{$substr:["$title",0,2]}}}])
{ "_id" : 8751, "new_title" : "Th" }
{ "_id" : 8752, "new_title" : "Di" }
{ "_id" : 8645, "new_title" : "Ec" }
{ "_id" : 7000, "new_title" : "Th" }
{ "_id" : 7020, "new_title" : "Il" }
> db.books.aggregate([{$project:{new_title:{$substrBytes:["$title",0,2]}}}])
{ "_id" : 8751, "new_title" : "Th" }
{ "_id" : 8752, "new_title" : "Di" }
{ "_id" : 8645, "new_title" : "Ec" }
{ "_id" : 7000, "new_title" : "Th" }
{ "_id" : 7020, "new_title" : "Il" }
> _
```

聚合管道操作符-$match

{ $match: { <query> } }

用于过滤文档（与find函数的query参数用法相同）

注意：如果$match位于管道中第一个阶段的话，可以借助索引加快查询，$match尽量出现在管道的前面，可以提早过滤文档，减少流经后续阶段的文档数量

例如：

```
db.articles.aggregate( [ {
 $match: { $or: [
 { score: { $gt: 70, $lt: 90 } },
 { views: { $gte: 1000 } }
] } }] );
```

聚合管道操作符-$limit

{ $limit: <positive integer> }

限制返回的文档数量

例如：

db.article.aggregate([{ $limit : 5 }]);

聚合管道操作符-$skip

{ $skip: <positive integer> }

跳过指定数量的文档

db.article.aggregate({ $skip : 5 });

聚合管道操作符-$sort

```
var results = db.stocks.aggregate( [ {
 $project : { cusip: 1, date: 1, price: 1, _id: 0 } },
{ $sort : { cusip : 1, date: 1 } } ],
{ allowDiskUse: true }
)
```

聚合管道操作符-$group

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

将集合中的文档进行分组

例如：

```
db.books.aggregate( [ { $group : { _id : "$author",
totalPublish:{$sum:”$publish”},
books: { $push: "$title" } } }
 ] )
```

聚合管道操作符-$sort

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

对输入的文档进行排序

例如：

db.users.aggregate([{ $sort : { age : -1, posts: 1 } }])

默认情况下，sort阶段只能在内存中进行，最大可使用100MB内存。

如果处理数据集比较大时，可以使用**allowDiskUse**选项。

聚合管道操作符-$unwind

{ $unwind: <field path> }，参数是数组类型

将文档按照数组字段拆分成多条文档，每条文档包含数组中的一个元素

New in version 3.2.

{ $unwind: {

path: <field path>,

includeArrayIndex: <string>,

preserveNullAndEmptyArrays: <boolean> } }

聚合管道操作符-$unwind

{ $unwind: <field path> }，参数是数组类型

将文档按照数组字段拆分成多条文档，每条文档包含数组中的一个元素

例如：{ "_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" }

聚合管道操作符-$group

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

将集合的文档进行分组，输出文档包含_id键，_id键包含了每个组的唯一key

_id为null,所有的文档被当做一个组

```
> db.books.aggregate([{ $group : { _id :"$author", books:{$push:"$title"}}}])
{ "_id" : "Homer", "books" : [ "The Odyssey", "Iliad" ] }
{ "_id" : "Dante", "books" : [ "The Banquet", "Divine Comedy", "Eclogues" ] }
> _
```

默认情况下，分组阶段只能在内存中进行，最大可使用100MB内存。如果处理数据集比较大时，可以使用**allowDiskUse**选项，数据写入硬盘。

聚合管道操作符-$group

按照年月日进行分组

```
db.sales.aggregate( [ {
$group:{ _id : { month: { $month: "$date" }, day: { $dayOfMonth: "$date" },
year: { $year: "$date" } },
 totalPrice: { $sum: { $multiply: [ "$price", "$quantity" ] } },
averageQuantity: { $avg: "$quantity" },
 count: { $sum: 1 } } } ] )
```

聚合管道操作符-$group

对数组进行分组

```
db.books.aggregate( [ {
$group : { _id : "$author",
books: { $push: "$title" } } } ] )
```

聚合管道操作符-$lookup

实现了join操作

{ $lookup: {

from: <collection to join>,

localField: <field from the input documents>,

foreignField: <field from the documents of the "from" collection>,

as: <output array field> } }

聚合管道操作符-$lookup

内连接的查询

```
db.orders.aggregate([ {
 $lookup: {
from: "inventory",
localField: "item",
foreignField: "sku",
as: "inventory_docs"
 } } ])
```

聚合管道操作符-$out

{ $out: "<output-collection>" }

将聚合结果存入集合中

例如：

```
db.books.aggregate( [ {
$group : { _id : "$author", books: { $push: "$title" } } },
 { $out : "authors" }
] )
```

聚合管道操作符-聚合管道表达式

阶段操作符可以看作是“键”，所对应的“值”称为管道表达式。管道表达式可以看做是管道操作符的操作数。

例如：

```
{ $group : { _id : "$author",
totalPublish:{$sum:”$publish”},
books: { $push: "$title" } } }
] )
```

管道表达式是一个文档结构、由字段名、字段值、和表达式操作符组成。

语法如下：

{<operator>:[<arg1>,<arg2>…]}或

{<operator>:<arg1>]}

聚合管道操作符-聚合管道表达式

操作符种类	作用	举例
布尔类型	且、或、非	$and $or $not
比较类型	等于、大于 、小于	$eq $gt $lt $ne $gte $cmp
算术功能	+-*/	$add $substract $mutiply $divide
字符串类型	字符串拼接、取子集、大小写	$concat $substr $toLower $toUpper
累加	最大、最小、平均值	$sum $avg $max $first $last
集合操作	集合的交、并、差运算	$setInterse $setUnion $setEquls
日期操作	取年、月、日、时、分、秒等	$year $month $hour $minute $week

本章大纲

- 聚合管道的概述
- 聚合管道操作符
- **聚合管道的使用**

聚合管道的使用

默认情况下，整个集合的所有文档作为聚合管道的输入，为了提高数据的处理效率可以使用下面的几个策略：

- 将$match和$sort放于管道开始阶段，这样可以利用集合建立的索引来提高文档的处理效率
- 提早过滤，在管道的初始阶段，可以使用$match，$limit以及$skip提早过滤，可以减少流经后续阶段的文档数量

聚合管道的使用

当聚合管道命令执行时，数据库本身也会对各个阶段自动进行优化，主要包括下面几种情况：

- $sort+$match顺序优化

如果$match阶段出现在$sort阶段之后，优化器会将$match移动到$sort的前面

- $skip+$limit顺序优化

如果$skip阶段出现在$limit阶段之后，优化器会将$limit移动到$skip的前面,移动后$limit参数的值等于原来的值加上skip参数的值

{$skip:1},{$limit:5}

优化后，{$limit:6},{$skip:1}

聚合管道的使用

使用聚合管道的限制如下：

- 返回结果大小

聚合返回的是一个文档，不能超过16MB，如果返回的结果是游标或者存储到集合中，不受16MB的限制。

- 内存

聚合管道的每个阶段最多只能占用100MB 内存，超过100MB时，会产生错误，如果需要处理大数据集，可以使用allowDiskUse选项

聚合管道的使用

单目（Single Purpose Aggregation Operations ）的聚合命令，常用的主要有两个：count和distinct

与聚合管道相比，功能单一，使用简单频繁。例如：

```
db.inventory.distinct( "dept" )
db.orders.count( { ord_dt: { $gt: new Date('01/01/2012') } } )
```

聚合管道的使用

- 下载测试数据 http://media.mongodb.org/zips.json
- 导入测试数据

mongoimport --db student --collection zipcodes --file zips.json

数据格式

{ “_id” : “01001”,//一个城市有多个邮编

"city" : "AGAWAM",

"loc" : [-72.622739, 42.070206],

“pop” : 15338,//地区人口数量

"state" : "MA" }

聚合管道的使用-练习

1、统计人口数量超过1000万的州

2、每个州城市拥有的平均人口数量

3、每个州的人口最多和最少的城市

4、统计所有的州名