MongoDB中聚合工具Aggregate等的介绍与使用

Aggregate是MongoDB提供的众多工具中的比较重要的一个，类似于SQL语句中的GROUP BY。聚合工具可以让开发人员直接使用MongoDB原生的命令操作数据库中的数据，并且按照要求进行聚合。

MongoDB提供了三种执行聚合的方法：Aggregation Pipleline，map-reduce功能和 Single Purpose Aggregation Operations

其中用来做聚合操作的几个函数是

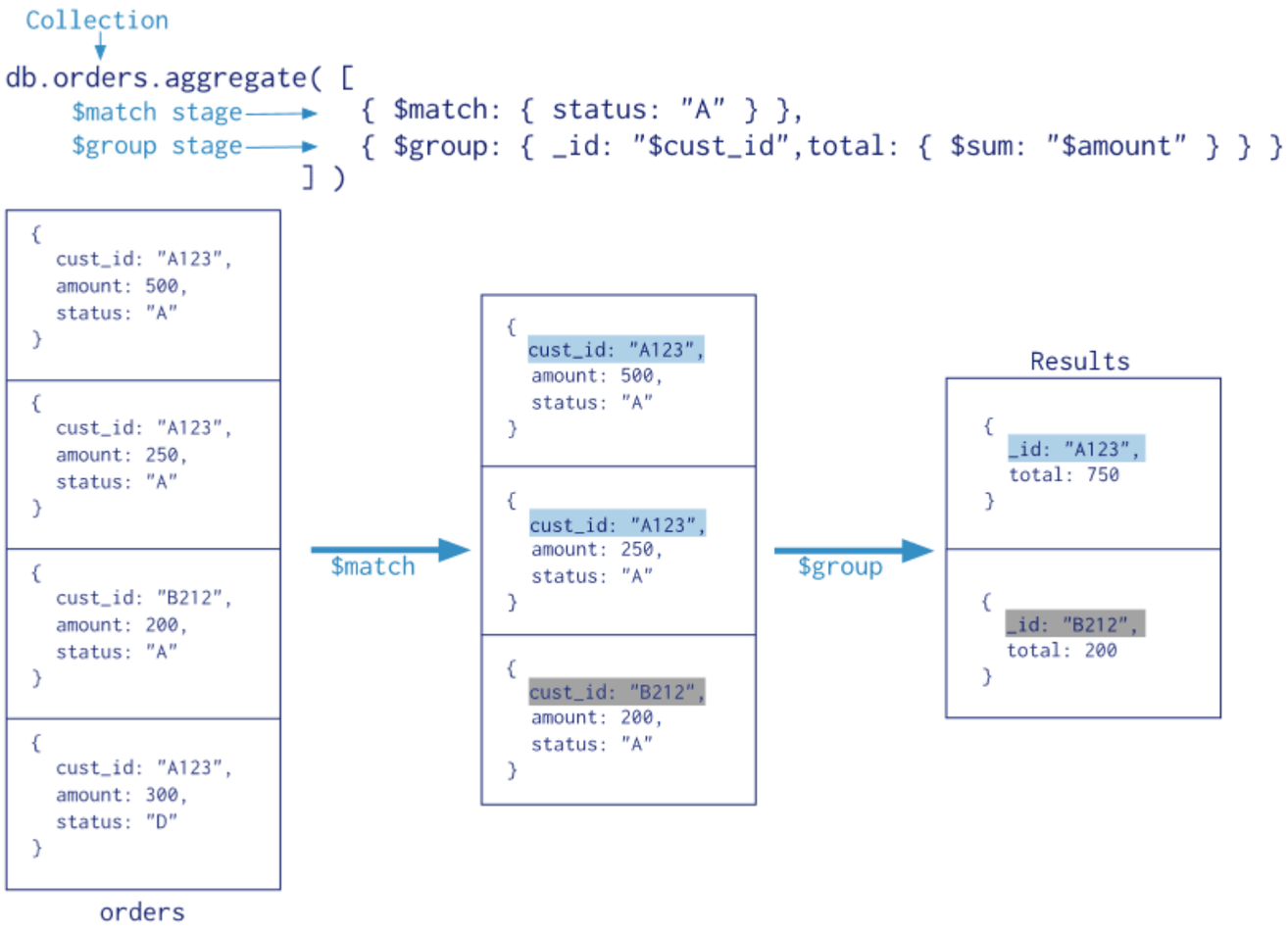
- aggregate(pipeline,options) 指定 group 的 keys, 通过操作符 $push/$addToSet/$sum 等实现简单的 reduce, 不支持函数/自定义变量
- group({ key, reduce, initial [, keyf] [, cond] [, finalize] }) 支持函数(keyf) mapReduce 的阉割版本
- mapReduce
- count(query)
- distinct(field,query)

1、Aggregation Pipleline

> MongoDB’ s aggregation framework is modeled on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into an aggregated result.

管道在*nix中将上一个命令输出的数据作为下一个命令的参数。MongoDB中的管道聚合非常实用，提供高效的数据聚合，并且是MongoDB中数据聚合的首选方法

官方给的图：




```
[
    {$match: {status: "A"}},
    {$group: {_id: "$cust_id", total: {$sum: "$amount"}}}
]
```

aggreagte是一个数组，其中包含多个对象（命令），通过遍历Pipleline数组对collection中的数据进行操作。

$match：查询条件

$group：聚合的配置

- _id代表你想聚合的数据的主键，上述数据中，你想聚合所有cust_id相同的条目的amount的总和，那_id即被设置为cust_id。_id为**必须**，你可以填写一个空值。
- total代表你最后想输出的数据之一，这里total是每条结果中amount的总和。
- $sum是一个聚合的操作符，另外的操作符你可以在官方文档中找到。上图中的命令表示对相同主键（_id）下的amount进行求和。如果你想要计算主键出现的次数，可以把命令写成如下的形式 {$sum: 1}

聚合的过程

看一下图例，所有的数据先经过$match命令，只留下了status为A的数据，接着，对筛选出的数据进行聚合操作，对相同cust_id的数据进行计算amount总和的操作，最后输出结果。

二、aggregate具体介绍

接受两个参数 pipeline/options, pipeline 是 array, 相同的 operator 可以多次使用

pipeline 支持的方法

- $geoNear geoNear命令可以在查询结果中返回每个点距离查询点的距离
- $group 指定 group 的 _id(key/keys) 和基于操作符($push/$sum/$addToSet/...) 的累加运算
- $limit 限制条件
- $match 输入过滤条件
- $out 将输出结果保存到 collection
- $project 修改数据流中的文档结构
- $redact 是 $project/$match 功能的合并
- $skip 跳过
- $sort 对结果排序
- $unwind 拆解数据

$group 允许用的累加操作符 $addToSet/$avg/$first/$last/$max/$min/$push/$sum，不被允许的累加操作符$each... ，默认最多可以用 100MB RAM, 增加allowDiskUse可以让$group操作更多的数据

下面是aggregate的用法

```
db.newtest.aggregate([
    {$match: {}},
    {$skip: 10}, // 跳过 collection 的前 10 行
    {$project: {group: 1, datetime: 1, category: 1, count: 1}},
    // 如果不选择 {count: 1} 最后的结果中 count_all/count_avg = 0
    {$redact: { // redact 简单用法 过滤 group != 'A' 的行
        $cond: [{$eq: ["$group", "A"]}, "$$DESCEND", "$$PRUNE"]
    }},
    {$group: {
        _id: {year: {$year: "$datetime"}, month: {$month: "$datetime"}, day: {$dayOfMonth: "$datetime"}},
        group_unique: {$addToSet: "$group"},
        category_first: {$first: "$category"},
        category_last: {$last: "$category"},
        count_all: {$sum: "$count"},
        count_avg: {$avg: "$count"},
        rows: {$sum: 1}
    }},
    // 拆分 group_unique 如果开启这个选项, 会导致 _id 重复而无法写入 out 指定的 collection, 除非再 $group 一次
    // {$unwind: "$group_unique"},
    // 只保留这两个字段
    {$project: {group_unique: 1, rows: 1}},
    // 结果按照 _id 排序
    {$sort: {"_id": 1}},
```

```
    // 只保留 50 条结果
    // {$limit: 50},
    // 结果另存
    {$out: "data_agg_out"},
], {
    explain: true,
    allowDiskUse: true,
    cursor: {batchSize: 0}
})
db.data_agg_out.find()
db.data_agg_out.aggregate([
    {$group: {
        _id: null,
        rows: {$sum: '$rows'}
    }}
])
db.data_agg_out.drop()
```

- $match 聚合前数据筛选
- $skip 跳过聚合前数据集的 n 行, 如果 {$skip: 10}, 最后 rows = 5000000 - 10
- $project 之选择需要的字段, 除了 _id 之外其他的字段的值只能为 1
- $redact 看了文档不明其实际使用场景, 这里只是简单筛选聚合前的数据
- $group 指定各字段的累加方法
- $unwind 拆分 array 字段的值, 这样会导致 _id 重复
- $project 可重复使用多次 最后用来过滤想要存储的字段
- $out 如果 $group/$project/$redact 的 _id 没有重复就不会报错
- 以上方法中 $project/$redact/$group/$unwind 可以使用多次

二、group

group 比 aggregate 好的一个地方是 map/reduce 都支持用 function 定义, 下面是支持的选项

- ns 如果用 db.runCommand({group: {}}) 方式调用, 需要 ns 指定 collection
- cond 聚合前筛选
- key 聚合的 key
- initial 初始化 累加 结果
- $reduce 接受 (curr, result) 参数, 将 curr 累加到 result
- keyf 代替 key 用函数生成聚合用的主键
- finalize 结果处理

需要保证输出结果小于 16MB 因为 group 没有提供转存选项

```
db.data.group({
    cond: {'group': 'A'},
    // key: {'group': 1, 'category': 1},
    keyf: function(doc) {
        var dt = new Date(doc.created);
        // or
        // var dt = doc.datetime;
        return {
            year: doc.datetime.getFullYear(),
            month: doc.datetime.getMonth() + 1,
            day: doc.datetime.getDate()
        }
    },
    initial: {count: 0, category: []},
    $reduce: function(curr, result) {
        result.count += curr.count;
        if (result.category.indexOf(curr.category) == -1) {
            result.category.push(curr.category);
        }
    },
    finalize: function(result) {
        result.category = result.category.join();
    }
})
```

如果要求聚合大量数据, 就需要用到 mapReduce

三、mapReduce

- query 聚合前筛选
- sort 对聚合前的数据排序 用来优化 reduce
- limit 限制进入 map 的数据
- map(function) emit(key, value) 在函数中指定聚合的 K/V
- reduce(function) 参数 (key, values) key 在 map 中定义了, values 是在这个 K 下的所有 V 数组
- finalize 处理最后结果
- out 结果转存 可以选择另外一个 db
- scope 设置全局变量
- jdMode(false) 是否(默认是)把 map/reduce 中间结果转为 BSON 格式, BSON 格式可以利用磁盘空间, 这样就可以处理大规模的数据集
- verbose(true) 详细信息

如果设 jsMode 为 true 不进行 BSON 转换, 可以优化 reduce 的执行速度, 但是由于内存限制最大在 emit 数量小于 500,000 时使用

写 mapReduce 时需要注意

- emit 返回的 value 必须和 reduce 返回的 value 结构一致
- `reduce` 函数必须幂等
- 详见 Troubleshoot the Reduce Function

```
db.data.mapReduce(function() {
    var d = this.datetime;
    var key = {
        year: d.getFullYear(),
        month: d.getMonth() + 1,
        day: d.getDate(),
    };
    var value = {
        count: this.count,
        rows: 1,
        groups: [this.group],
    }
    emit(key, value);
}, function(key, vals) {
    var reducedVal = {
        count: 0,
        groups: [],
        rows: 0,
    };
    for(var i = 0; i < vals.length; i++) {
        var v = vals[i];
        reducedVal.count += v.count;
        reducedVal.rows += v.rows;
        for(var j = 0; j < v.groups.length; j ++) {
            if (reducedVal.groups.indexOf(v.groups[j]) == -1) {
                reducedVal.groups.push(v.groups[j]);
            }
        }
    }
    return reducedVal;
}, {
    query: {},
    sort: {datetime: 1},    // 需要索引 否则结果返回空
    limit: 50000,
    finalize: function(key, reducedVal) {
        reducedVal.avg = reducedVal.count / reducedVal.rows;
        return reducedVal;
    },
    out: {
        inline: 1,
        // replace: "",
        // merge: "",
        // reduce: "",
```

```
    },
    scope: {},
    jsMode: true
})
```

测试数据：

```
> db.newtest.find()
{ "_id" : ObjectId("5a2544352ba57ccba824d7bf"), "group" : "E", "created" : 1402764223, "count" : 63, "datetime" : 1512391126, "title" : "aa", "category" : "C8" }
{ "_id" : ObjectId("5a2544512ba57ccba824d7c0"), "group" : "I", "created" : 1413086660, "count" : 93, "datetime" : 1512391261, "title" : "bb", "category" : "C10" }
{ "_id" : ObjectId("5a2544562ba57ccba824d7c1"), "group" : "H", "created" : 1440750343, "count" : 41, "datetime" : 1512391111, "title" : "cc", "category" : "C1" }
{ "_id" : ObjectId("5a2544562ba57ccba824d7c2"), "group" : "S", "created" : 1437710373, "count" : 14, "datetime" : 1512392136, "title" : "dd", "category" : "C10" }
{ "_id" : ObjectId("5a2544562ba57ccba824d7c3"), "group" : "Z", "created" : 1428307315, "count" : 78, "datetime" : 1512391166, "title" : "ee", "category" : "C5" }
{ "_id" : ObjectId("5a2544562ba57ccba824d7c4"), "group" : "R", "created" : 1402809274, "count" : 74, "datetime" : 1512391162, "title" : "ff", "category" : "C9" }
{ "_id" : ObjectId("5a2544562ba57ccba824d7c5"), "group" : "Y", "created" : 1400571321, "count" : 66, "datetime" : 1512139164, "title" : "gg", "category" : "C2" }
{ "_id" : ObjectId("5a2544562ba57ccba824d7c6"), "group" : "L", "created" : 1416562128, "count" : 5, "datetime" : 1512393165, "title" : "hh", "category" : "C1" }
{ "_id" : ObjectId("5a2544562ba57ccba824d7c7"), "group" : "E", "created" : 1414057884, "count" : 12, "datetime" : 1512391165, "title" : "ii", "category" : "C3" }
{ "_id" : ObjectId("5a2544572ba57ccba824d7c8"), "group" : "L", "created" : 1418879346, "count" : 67, "datetime" : 1512391167, "title" : "gg", "category" : "C3" }
```

四、总结

method	allowDiskUse	out	function
aggregate	true	pipeline/collection	false
group	false	pipeline	true
mapReduce	jsMode	pipeline/collection	true

- `aggregate` 基于累加操作的的聚合 可以重复利用 `$project`/`$group` 一层一层聚合数据, 可以用于大量数据(单输出结果小于 16MB) 不可用于分片数据
- `mapReduce` 可以处理超大数据集 需要严格遵守 mapReduce 中的结构一致/幂等 写法, 可增量输出/合并, 见 `out` options
- `group` RDB 中的 `group by` 简单需求可用(只有 inline 输出) 会产生 `read lock`

分类: Mongodb

标签： mongo， mongodb， aggregate

好文要顶 关注我 收藏该文

踏雪无痕SS
关注 - 1
粉丝 - 96
+加关注

2 0

« 上一篇：MongoDB中MapReduce介绍与使用
» 下一篇：MongoDB优化与一些需要注意的细节

posted @ 2017-12-04 21:16 踏雪无痕SS 阅读(341) 评论(1) 编辑 收藏

评论列表

#1楼 2018-02-12 17:48 David Smith gf

觉得SQL to Aggregation Mapping Chart 更全更有用。

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

最新IT新闻:
· 俞敏洪：不需要有个清晰的梦想才去做，需要的是每一天都要前行
· 纳德拉：Office 365的增长机会大过公司历史上任何产品
· 三星官宣Galaxy Note 9：预装Bixby2.0 9月发布
· 微软量子开发套件更新 现已支持macOS和Linux
· 谷歌旗下DeepMind教AI预测死亡
» 更多新闻...

最新知识库文章:
· 和程序员谈恋爱
· 学会学习
· 优秀技术人的管理陷阱
· 作为一个程序员，数学对你到底有多重要
· 领域驱动设计在互联网业务开发中的实践
» 更多知识库文章...

历史上的今天:
2015-12-04 大型web系统数据缓存设计