1.MongoDB聚合操作
2.MongoDB聚合优化，
3.MongoDB索引原理及实战

MongoDB聚合框架是一个计算框架作用在一个或几个集合对集合中的数据进行一系列运算
将这些数据转化为期望的形式

如： 现在有一组数据

```
db.orders.insertMany(
[
{
 zip:"000001",
 phone:"13101010101",
 name:"LiuBei",
 status:"created",
 shippingFee:10,
 orderLines:[
  {product:"Huawei Meta30 Pro",sku:"2002",qty:100,price:6000,cost:5599},
  {product:"Huawei Meta40 Pro",sku:"2003",qty:10,price:7000,cost:6599},
  {product:"Huawei Meta40 5G",sku:"2004",qty:80,price:4000,cost:3700}
  ]
},

{
 zip:"000001",
 phone:"13101010101",
 name:"LiuBei",
 status:"created",
 shippingFee:10,
 orderLines:[
 {product:"Huawei Meta30 Pro",sku:"2002",qty:100,price:6000,cost:5599},
 {product:"Huawei Meta40 Pro",sku:"2003",qty:10,price:7000,cost:6599},
```

```
  {product:"Huawei Meta40 5G",sku:"2004",qty:80,price:4000,cost:3700}
  ]
},

{
 zip:"000001",
 phone:"13101010101",
 name:"LiuBei",
 status:"created",
 shippingFee:10,
 orderLines:[
 {product:"Huawei Meta30 Pro",sku:"2002",qty:100,price:6000,cost:5599},
 {product:"Huawei Meta40 Pro",sku:"2003",qty:10,price:7000,cost:6599},
 {product:"Huawei Meta40 5G",sku:"2004",qty:80,price:4000,cost:3700}
 ]
}]
);
```

原价总额，订单总额

添加两个字段，值为每个订单的原价总价和 订单总额

```
db.orders.aggregate([{$addFields: {
 totalPrice:{ $sum: "$orderLines.price"},
 totalCost: { $sum: "$orderLines.cost"},
}}]);
```

```
"shippingFee" : 10,
"orderLines" : [
        {
                "product" : "Huawei Meta30 Pro",
                "sku" : "2002",
                "qty" : 100,
                "price" : 6000,
                "cost" : 5599
        },
        {
                "product" : "Huawei Meta40 Pro",
                "sku" : "2003",
                "qty" : 10,
                "price" : 7000,
                "cost" : 6599
        },
        {
                "product" : "Huawei Meta40 5G",
                "sku" : "2004",
                "qty" : 80,
                "price" : 4000,
                "cost" : 3700
        }
],
"totalPrice" : 17000,
"totalCost" : 15898
```

```
按 totalPrice 进行排序
[{
 $addFields: {
 totalPrice: {
 $sum: "$orderLines.price"
 },
 totalCost: {
 $sum: "$orderLines.cost"
 },
  }
}
, // stage 1

 {
 $sort: {
 totalPrice: -1
 }
} // stage 2
]

```

第一个阶段：

针对整个集合添加两个字段：
totalPrice: 订单原价总额
totalCost：订单实际总额

并将结果传到第二个阶段

第二个阶段：

按订单总价进行排序

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

聚合管道阶段

$project 对输入文档进行再次投影
$match 对输入文档进行筛选
$limit 筛选出管道内前 N 篇文档
$skip 跳过管道内前N篇文档
$unwind 展开输入文档中的数组字段
$sort 对输入文档进行排序
$lookup 对输入文档进行查询操作
$group 对输入文档进行分组
$out 对管道中的文档输出

准备数据

```
db.userInfo.insertMany(
 [
 {nickName:"zhangsan",age:18},
 {nickName:"lisi",age:20}]
 );
```

聚合管道操作

$project： 投影操作， 将原始字段投影成指定名称， 如将 集合中的 nickName 投影成 name

```
db.userInfo.aggregate({$project:{ name:"$nickName"}});
```

$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.userInfo.aggregate({$project:{ name:"$nickName",_id:0,age:1}});
```

$match
$match 进行文档筛选

```
db.userInfo.aggregate({$match:{ nickName:"lisi"}});
```

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

将筛选 和 投影结合使用

```

 db.userInfo.aggregate(
 [
 { $match: { nickName: "lisi"}},
 { $project: { _id:0, name: "$nickName", age:1}}
]
);
```

```
db.userInfo.aggregate(
[
 { $match:
 { $and:
 [ { age: { $gte:20}},
 { nickName:{ $eq:"lisi"}}
 ]
 }
 },
  {
  $project: { _id:0, name: "$nickName", age:1}
}
]
);
```

$limit

$skip

```
db.userInfo.aggregate({$limit:1});
db.userInfo.aggregate({$skip:1});
```

 $unwind

将数组打平

构造数据

```
db.userInfo.insertOne(
{ "nickName" : "xixi",
 "age" : 35,
 "tags" : ["80","IT","BeiJing"]
}
);
```

```
db.userInfo.aggregate(
{ $unwind:{path:"$tags"}}
);
```

includeArrayIndex: 加上数组元素的索引值， 赋值给后面指定的字段

```
db.userInfo.aggregate(
 {$unwind: {path: "$tags",includeArrayIndex:"arrIndex"}}
);
```

preserveNullAndEmptyArrays:true

```
 db.userInfo.aggregate(
 { $unwind:
 {
 path: "$tags",
 includeArrayIndex:"arrIndex",
 preserveNullAndEmptyArrays:true}
 }
);
```

展开时保留空数组，或者不存在数组字段的文档

$sort 对文档进行排序： 1 正序， -1 倒序

```
db.userInfo.aggregate({$sort:{age:-1}});
```

$lookup

使用单一字段值进行查询

```
$lookup:{
from: 需要关联的文档,
localField: 本地字段,
foreignField: 外部文档关联字段,
as 作为新的字段，添加到文档中
}
```

```
db.account.insertMany(
 [
 {_id:1,name:"zhangsan",age:19},
 {_id:2,name:"lisi",age:20}
]
);
```

```
db.accountDetail.insertMany(
[
{aid:1,address:["address1","address2"]}
]
);
```

```
db.accountDetail.aggregate(
{
 $lookup:
 {
 from:"account",
 localField:"aid",
 foreignField:"_id",
 as: "field1"
 }
 }
);
```

$group

```
$group:{
 _id: 对哪个字段进行分组,
field1:{ accumulator1: expression1 }
}
```

group 聚合操作默认不会对输出结果进行排序

对于group，聚合操作主要有以下几种

$addToSet： 将分组中的元素添加到一个数组中，并且自动去重
$avg 返回分组中的平均值， 非数值直接忽略
$first 返回分组中的第一个元素
$last 返回分组中的最后一个元素
$max 返回分组中的最大元素
$min 回分组中的最小元素
$push 创建新的数组，将值添加进去
$sum 求分组数值元素和

构造一组数据

```
db.sales.insertMany(
[
{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : ISODate("2014-01-01T08:00:00Z") },
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "date" : ISODate("2014-02-03T09:00:00Z") },
{ "_id" : 3, "item" : "xyz", "price" : 5, "quantity" : 5, "date" : ISODate("2014-02-03T09:05:00Z") },
{ "_id" : 4, "item" : "abc", "price" : 10, "quantity" : 10, "date" : ISODate("2014-02-15T08:00:00Z") },
{ "_id" : 5, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-02-15T09:12:00Z") },
{ "_id" : 6, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-02-15T09:12:00Z") }
]
);
```

$addToSet

查看每天，卖出哪几种商品项目
按每天分组， 将商品加入到去重数组中

```
db.sales.aggregate(
 [
 {
 $group:
 {
 _id: { day: { $dayOfYear: "$date"}, year: { $year: "$date" } },
```

```
itemsSold: { $addToSet: "$item" }
}
}
 ]
)
```

$avg: 求数值的平均值

```
db.sales.aggregate(
 [
 {
 $group:
 {
 _id: "$item",
 avgAmount: { $avg: { $multiply: [ "$price", "$quantity" ] } },
 avgQuantity: { $avg: "$quantity" }
 }
 }
 ]
)
```

$push

创建新的数组，存储，每个分组中元素的信息

```
db.sales.aggregate(
 [
 {
 $group:
 {
 _id: { day: { $dayOfYear: "$date"}, year: { $year: "$date" } },
 itemsSold: { $push: { item: "$item", quantity: "$quantity" } }
 }
 }
 ]
)

```

group 阶段有 100m内存的使用限制， 默认情况下，如果超过这个限制会直接返回 error，

可以通过设置 allowDiskUse 为 true 来避免异常， allowDiskUse 为 true 将利用临时文件来辅助实现group操作。

$out
将聚合结果写入另一个文档

```
db.sales.aggregate(
 [
 {
 $group:
 {
 _id: { day: { $dayOfYear: "$date"}, year: { $year: "$date" } },
 itemsSold: { $push: { item: "$item", quantity: "$quantity" } }
 }
 },
  { $out:"output"}
  ]
)
```

更多聚合操作参考： https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/

管道优化

1. 投影优化
聚合管道可以确定它是否仅需要文档中的字段的子集来获得结果。如果是这样，管道将只使用那些必需的字段，减少通过管道的数据量。

2. 管道符号执行顺序优化
对于包含投影阶段($project或$unset或$addFields或$set)后跟$match阶段的聚合管道，MongoDB 将$match阶段中不需要在投影阶段计算的值的任何过滤器移动到投影前的新$match阶段

3. $sort + $match
如果序列中带有$sort后跟$match，则$match会移动到$sort之前，以最大程度的减少要排序的对象的数量

4. $project/ $unset + $skip序列优化
当有一个$project或$unset之后跟有$skip序列时，$skip 会移至$project之前。

5.$limit+ $limit合并
当$limit紧接着另一个时 $limit，两个阶段可以合并为一个阶段 $limit，其中限制量为两个初始限制量中的较小者。

6. skip+ $skip 合并

当$skip紧跟另一个$skip，这两个阶段可合并成一个单一的$skip，其中跳过量为总和的两个初始跳过量。

7. $match+ $match合并
当一个$match紧随另一个紧随其后时 $match，这两个阶段可以合并为一个单独 $match的条件 $and

```
{ $match: { year: 2014 } },
{ $match: { status: "A" } }
```

优化后

```
{ $match: { $and: [ { "year" : 2014 }, { "status" : "A" } ] } }
```

索引

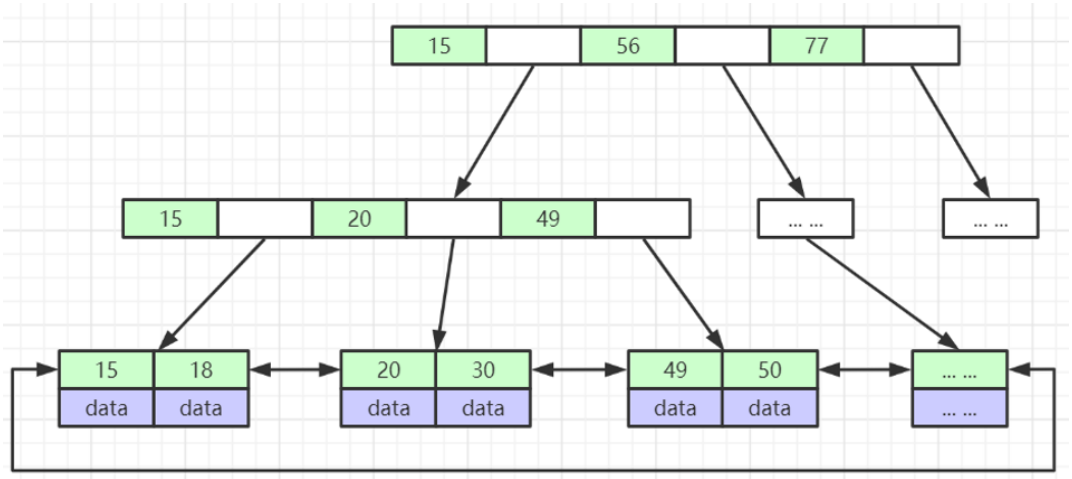
什么是索引?

索引是特殊的数据结构，它以一种易于遍历的形式存储集合数据集的一小部分。索引存储一个或一组特定字段的值，按字段的值排序。索引项的排序支持有效的**相等匹配**和基于**范围的查询**操作。此外，MongoDB可以通过使用索引中的排序返回**排序后**的结果。

WiredTiger: B+
比如基于 **B+ tree** 的索引数据结构图示如下：

单键索引

如基于主键ID 进行的B+ tree 数据结构



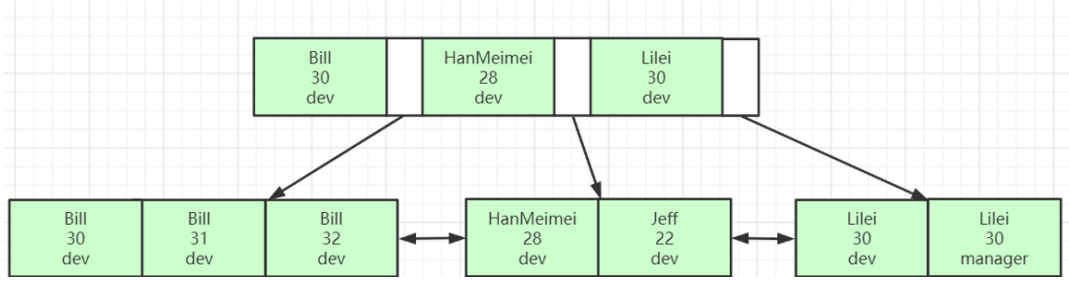
复合索引（复合索引只能支持前缀子查询）

(A,B,C)

基于name， age， position 建立的复合索引

name
name age
name age postion

age :
position
age+ position



索引的特点?

索引支持更快的查询
更快的排序

默认id索引

在创建集合期间，MongoDB 在_id字段上创建唯一索引。该索引可防止客户端插入两个具有相同值的文档。你不能将_id字段上的index删除。

创建索引:
db.collection.createIndex(<keys>, <options>)
<keys> 指定了创建索引的字段

构造一组数据

```
db.members.insertMany(
[
 {name:"zhangsan",age:19,tags:["00","It","SH"]},
 {name:"lisi",age:35,tags:["80","It","BJ"]},
 {name:"wangwu",age:31,tags:["90","It","SZ"]}
]
);
```

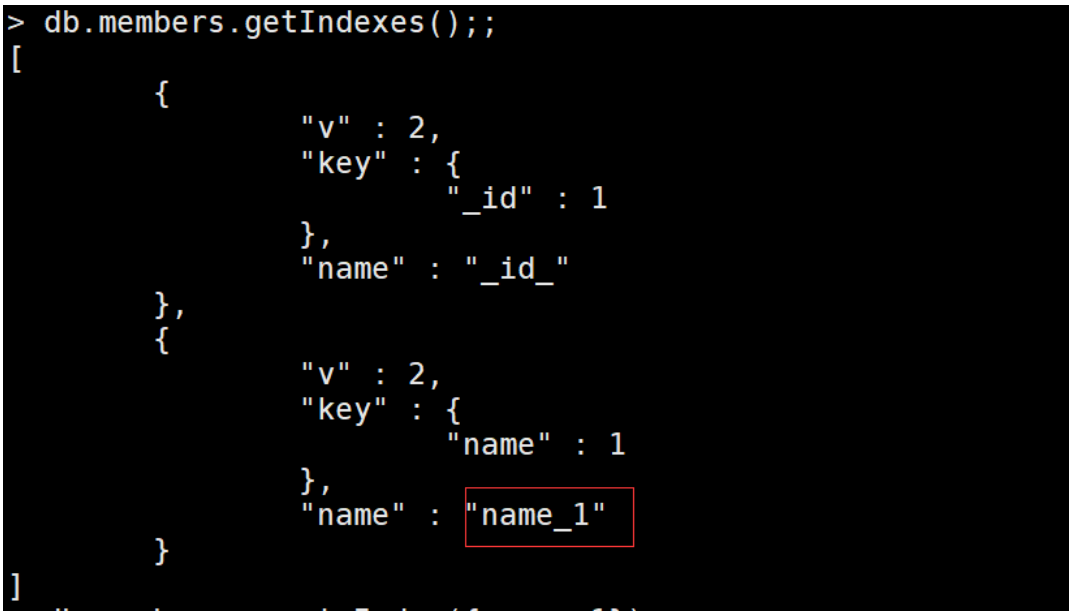
创建一个单键索引

```
db.members.createIndex({name:1});
```

索引的默认名称是索引键和索引中每个键的方向(即1或-1)的连接，使用下划线作为分隔符， 也可以通过指定 name 来自定义索引名称;

```
db.members.createIndex({name:1}, { name: "whatever u like."});
```

对于单字段索引和排序操作，索引键的排序顺序(升序或降序)并不重要，因为MongoDB可以从任何方向遍历索引。



查询集合中已经存在的索引

```
db.members.getIndexes();
```

创建一个复合索引

MongoDB支持在多个字段上创建用户定义索引，即 复合索引。
复合索引中列出的字段的顺序具有重要意义。如果一个复合索引由 {name: 1, age: -1} 组成，索引首先按 name 升序排序，然后在每个name值内按 age 降序 排序。

```
db.members.createIndex({ name:1,age:-1});
```

对于复合索引和排序操作，索引键的排序顺序(升序或降序)可以决定索引是否支持排序操作。

创建多键索引

MongoDB使用多键索引来索引存储在数组中的内容。如果索引包含数组值的字段，MongoDB为数组的每个元素创建单独的索引项。数组字段中的每一个元素，都会在多键索引中创建一个键

```
db.members.createIndex( { tags:1});
```

索引的效果解析

```
db.collection.explain().method(?)
```

可以使用 explain 进行分析的操作包含 aggregate, count, distinct, find ,group, remove, update
winningPlan： stage 的值含义

COLLSCAN: 整个集合扫描
IXScan: 索引扫描
FETCH: 根据索引指向的文档的地址进行查询
SORT: 需要再内存中排序，效率不高

覆盖查询

当查询条件和查询的<投影>只包含索引字段时，MongoDB直接从索引返回结果，而不扫描任何文档或将文档带入内存。这些覆盖的查询可能非常高效。

```
db.members.explain().find({ name:"zhangsan"},{_id:0, name:1});
```

这时不需要 fetch, 可以直接从索引中获取数据。

```
db.members.explain().find().sort( {name:1 ,age:-1}) ;
```

使用已创建索引的字段进行排序，能利用索引的顺序，不需要重新排序，效率高

```
 db.members.explain().find().sort( {name:1 ,age: 1}) ;
```

使用未创建索引的字段进行排序， 因为和创建索引时的顺序不一致，所以需要重新排序，效率低

如果需要更改某些字段上已经创建的索引，必须首先删除原有索引，再重新创建新索引，否则，新索引不会包含原有文档

```
db.collection.dropIndex()
```

使用索引名称删除索引

```
db.collection.dropIndex("name_1");
```

使用索引定义删除索引

```
db.collection.dropIndex({name:1,age:-1});
```

```
db.collection.createIndex(<keys>, <options>)
```

<options> 定义了创建索引时可以使用的一些参数，也可以指定索引的特性

索引的唯一性

索引的unique属性使MongoDB拒绝索引字段的重复值。除了唯一性约束，唯一索引和MongoDB其他索引功能上是一致的

```
db.members.createIndex({age:1},{unique:true});
```

如果文档中的字段已经出现了重复值，则不可以创建该字段的唯一性索引

如果新增的文档不具备加了唯一索引的字段，则只有第一个缺失该字段的文档可以被添加，索引中该键值被置为null。

复合键索引也可以具有唯一性，这种情况下，不同的文档之间，其所包含的复合键字段值的组合不可以重复。

索引的稀疏性

索引的稀疏属性可确保索引仅包含具有索引字段的文档的条目。索引会跳过没有索引字段的文档。
可以将稀疏索引与唯一索引结合使用，**以防止插入索引字段值重复的文档**，并跳过索引缺少索引字段的文档。

准备数据：

```
db.sparsedemo.insertMany([{name:"xxx",age:19},{name:"zs",age:20}]);
```

创建 唯一键，稀疏索引

```
db.sparsedemo.createIndex({name:1},{unique:true ,sparse:true});
```

如果同一个索引既具有唯一性，又具有稀疏性，就可以保存多篇缺失索引键值的文档了

```
db.sparsedemo.insertOne({name:"zs2w",age:20});
```

```
db.sparsedemo.insertOne({name:"zs2w2",age:20});
```

说明：如果只单纯的 唯一键索引，则 缺失索引键的字段，只能有一个

复合键索引也可以具有稀疏性，在这种情况下，只有在缺失复合键所包含的所有字段的情况下，文档才不会被加入到索引中。

索引的生存时间

针对日期字段，或者包含了日期元素的数组字段，可以使用设定了生存时间的索引，来自动删除字段值超过生存时间的文档。

构造数据：

```
db.members.insertMany( [

 {
 name:"zhangsanss",
 age:19,tags:["00","It","SH"],
 create_time:new Date()}
 ] );

```

```
db.members.createIndex({ create_time: 1},{expireAfterSeconds:30 });
```

在create_time字段上面创建了一个生存时间是30s的索引

复合键索引不具备生存时间的特性

当索引键是包含日期元素的数组字段时，数组中最小的日期将被用来计算文档是否已经过期
数据库使用一个后台线程来监测和删除过期的文档，删除操作可能会有一定的延迟

https://docs.mongodb.com/manual/core/index-compound/#index-ascending-and-descending
https://docs.mongodb.com/manual/reference/operator/aggregation/addFields/#pipe._S_addFields

文档：VIP-02 MongoDB聚合操作，索引.note
链接：http://note.youdao.com/noteshare?id=50fdb657a9b06950fa255a82555b44a6&sub=DEADCFA5FCE148C4BBD8D840EA69E0D3