概要:

1. mongoDB的聚合操作
2. mongodb 集群：复制
3. mongodb 集群：分片

一、 mongoDB的聚合操作

知识点:

1. pipeline 聚合
2. mapRedurce 聚合
3. 在聚合中使用索引

1.pipeline 聚合

pipeline相关运算符:

- $match ： 匹配过滤聚合的数据
- $project：返回需要聚合的字段
- $group：统计聚合数据

示例:

```
# $match 与 $project使用
db.emp.aggregate(
{$match:{"dep":{$eq:"客服部"}}},
{$project:{name:1,dep:1,salary:1}}
);

# $group 与 $sum 使用
db.emp.aggregate(
 {$project:{dep:1,salary:1}},
 {$group:{"_id":"$dep",total:{$sum:"$salary"}}}
 );

#  低于4000 忽略
db.emp.aggregate(
 {$match:{salary:{$gt:4000}}},
 {$project:{dep:1,salary:1}},
 {$group:{"_id":"$dep",total:{$sum:"$salary"}}}
 );

# 基于多个字段 进行组合group  部门+职位进行统计
db.emp.aggregate(
 {$project:{dep:1,job:1,salary:1}},
 {$group:{"_id":{"dep":"$dep","job":"$job"},total:{$sum:"$salary"}}}
 );
```

二次过滤

```
db.emp.aggregate(
 {$project:{dep:1,job:1,salary:1}},
 {$group:{"_id":{"dep":"$dep","job":"$job"},total:{$sum:"$salary"}}},
 {$match:{"$total":{$gt:10000}}}
 );
```

2.mapRedurce 聚合

mapRedurce 说明：　为什么需要 MapReduce?　(1) 海量数据在单机上处理因为硬件资源限制，无法胜任 (2) 而一旦将单机版程序扩展到集群来分布式运行，将极大增加程序的复杂度和开发难度　(3) 引入 MapReduce 框架后，开发人员可以将绝大部分工作集中在业务逻辑的开发上，而将 分布式计算中的复杂性交由框架来处理

mongodb中mapRedurce的使用流程

1. 创建Map函数，
2. 创建Redurce函数
3. 将map、Redurce 函数添加至集合中，并返回新的结果集
4. 查询新的结果集

示例操作

```
// 创建map 对象
var map1=function (){
    emit(this.job,1);
 }
// 创建reduce 对象
 var reduce1=function(job,count){
 return Array.sum(count);
 }
 // 执行mapReduce 任务 并将结果放到新的集合 result 当中
db.emp.mapReduce(map1,reduce1,{out:"result"})
// 查询新的集合
db.result.find()
```

```
# 使用复合对象作为key
 var map2=function (){
    emit({"job":this.job,"dep":this.dep},1);
 }

 var reduce2=function(key,values){
    return values.length;
 }

db.emp.mapReduce(map2,reduce2,{out:"result2"}).find()
```

mapRedurce的原理 在map函数中使用emit函数添加指定的 key 与Value ， 相同的key 将会发给Redurce进行聚合操作，所以Redurce函数中第二个参数 就是 所有集的数组。return 的显示就是聚合要显示的值。

3.在聚合中使用索引

通过$Math内 可以包合对$text 的运算 示例:

```
db.project.aggregate(
            {$match:{$text:{$search:"apache"}}},
            {$project:{"name":1,"price":1}},
            {$group:{_id:"$name",price:{$sum:"$price"}}}
)
```

关于索引 除了全文索引之外，还有单键索引。即整个字段的值作为索引。单键索引用值1和-1表示，分别代表正序和降序索引。

示例： de 创建单键索引

```
db.emp.createIndex({"dep":1})
```

查看基于索引的执行计划

```
db.emp.find({"dep":"客服部"}).explain()
```

除了单键索引外还可以创建联合索引如下:

```
db.emp.createIndex({"dep":1,"job":-1})
```

查看 复合索引的执行计划

```
db.emp.find({"dep":"ddd"}).explain()
```

查看索引在排序当中的使用

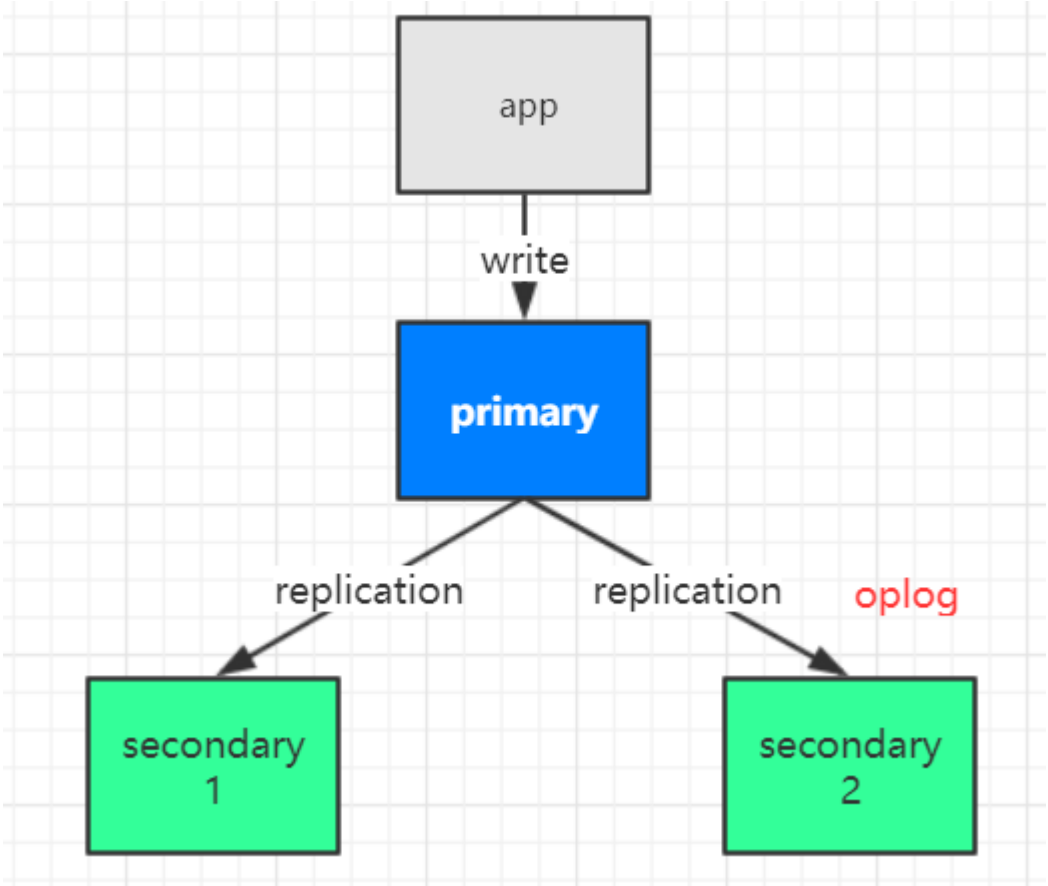
```
db.emp.find().sort({"job":-1,"dep":1}).explain()
```

二、mongodb 的主从复制机制

知识点:

1. 复制集群的架构
2. 复制集群搭建
3. 复制集群的选举配置

1.复制集群的架构



2.复制集群搭建基础示例

主节点配置

```
dbpath=/data/mongo/master
port=27017
fork=true
logpath=master.log
replSet=tulingCluster
```

从节点配置

```
dbpath=/data/mongo/slave
port=27018
fork=true
logpath=slave.log
replSet=tulingCluster
```

#子节点配置2

```
dbpath=/data/mongo/slave2
port=27019
fork=true
logpath=slave2.log
replSet=tulingCluster
```

- [ ] 分别启动三个节点
- [ ] 进入其中一个节点

集群复制配置管理

```
#查看复制集群的帮助方法
rs.help()
```

添加配置

```
// 声明配置变量
var cfg ={"_id":"tulingCluster",
            "members":[
                {"_id":0,"host":"127.0.0.1:27017"},
                {"_id":1,"host":"127.0.0.1:27018"}
            ]
         }
// 初始化配置
rs.initiate(cfg)
// 查看集群状态
rs.status()
```

变更节点示例：

```
// 插入新的复制节点
rs.add("127.0.0.1:27019")
// 删除slave 节点
rs.remove("127.0.0.1:27019")
```

注：默认节点下从节点不能读取数据。调用 rs.slaveOk() 解决。

3.复制集群选举操作

为了保证高可用，在集群当中如果主节点挂掉后，会自动 在从节点中选举一个 重新做为主节点。

选举的原理： 在mongodb 中通过在 集群配置中的 rs.属性值大小来决定选举谁做为主节点，通时也可以设置 arbiterOnly 为true 表示 做为裁判节点用于执行选举操作，该配置下的节点 永远不会被选举为主节点和从节点。

示例：

```
重新配置节点
var cfg ={"_id":"tulingCluster",
          "protocolVersion" : 1,
          "members":[
                {"_id":0,"host":"127.0.0.1:27017","priority":10},
                {"_id":1,"host":"127.0.0.1:27018","priority":2},
                {"_id":2,"host":"127.0.0.1:27019","arbiterOnly":true}
            ]
         }
// 重新装载配置，并重新生成集群节点。
rs.reconfig(cfg)
```

```
//重新查看集群状态
rs.status()
```

节点说明： PRIMARY 节点： 可以查询和新增数据 SECONDARY 节点：只能查询 不能新增 基于priority 权重可以被选为主节点 RBITER 节点： 不能查询数据 和新增数据 ， 不能变成主节点

三、mongodb 中的分片机制

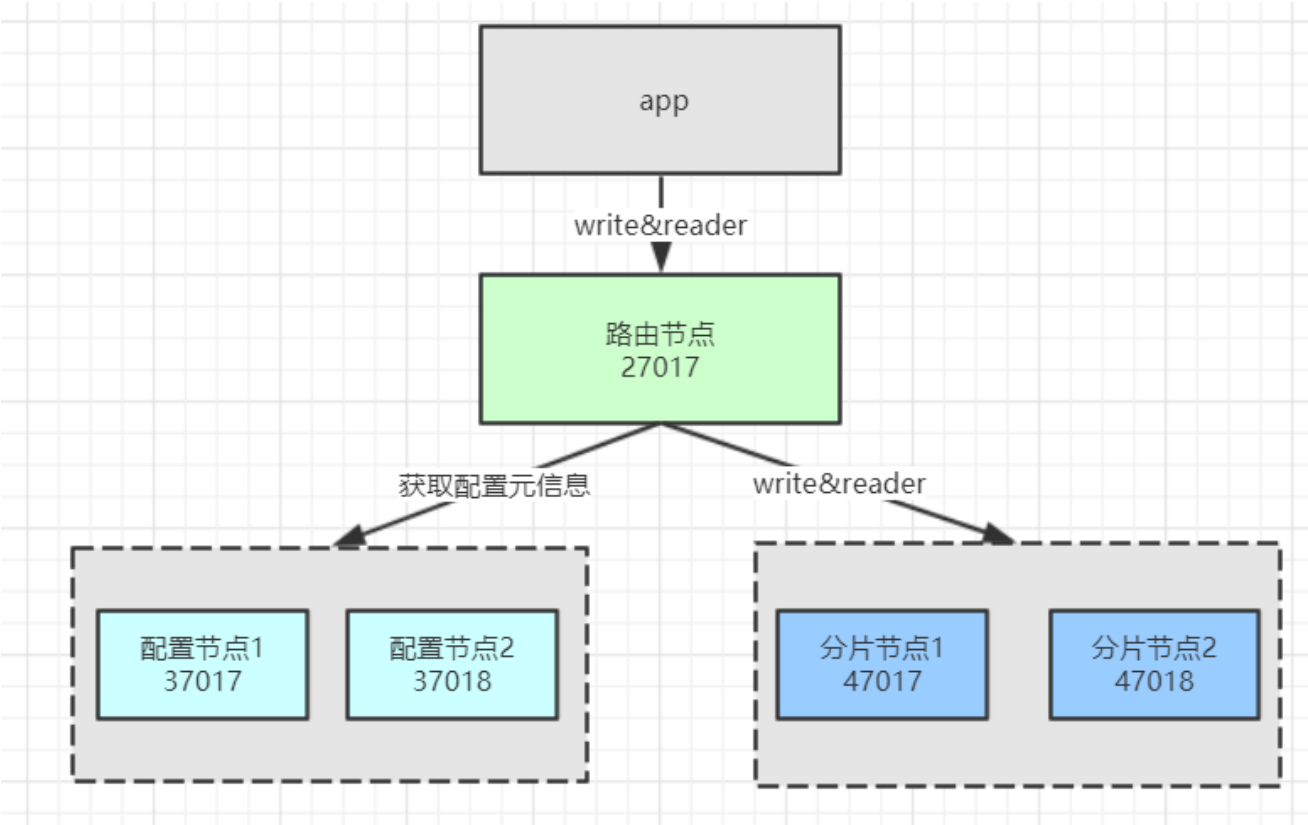
知识点：

1. 分片的概念
2. mongodb 中的分片架构
3. 分片示例

1.为什么需要分片?

随着数据的增长，单机实例的瓶颈是很明显的。可以通过复制的机制应对压力，但mongodb中单个集群的 节点数量限制到了12个以内，所以需要通过分片进一步横向扩展。此外分片也可节约磁盘的存储。

1.mongodb 中的分片架构



分片中的节点说明：

- 路由节点(mongos)：用于分发用户的请求，起到反向代理的作用。

- 配置节点(config)：用于存储分片的元数据信息，路由节基于元数据信息 决定把请求发给哪个分片。（3.4版本之后，该节点，必须使用复制集。）
- 分片节点(shard):用于实际存储的节点，其每个数据块默认为64M，满了之后就会产生新的数据库。

2.分片示例流程：

1. 配置 并启动config 节点集群
2. 配置集群信息
3. 配置并启动2个shard 节点
4. 配置并启动路由节点
5. 添加shard 节点
6. 添加shard 数据库
7. 添加shard 集合
8. 插入测试数据
9. 检查数据的分布
10. 插入大批量数据查看shard 分布
11. 设置shard 数据块为一M
12. 插入10万条数据

配置 并启动config 节点集群

节点1 config1-37017.conf

```
dbpath=/data/mongo/config1
port=37017
fork=true
logpath=logs/config1.log
replSet=configCluster
configsvr=true
```

节点2 config2-37018.conf

```
dbpath=/data/mongo/config2
port=37018
fork=true
logpath=logs/config2.log
replSet=configCluster
configsvr=true
```

进入shell 并添加 config 集群配置：

```
var cfg ={"_id":"configCluster",
          "protocolVersion" : 1,
          "members":[
                {"_id":0,"host":"127.0.0.1:37017"},
                {"_id":1,"host":"127.0.0.1:37018"}
            ]
         }
// 重新装载配置，并重新生成集群。
rs.initiate(cfg)
```

配置 shard 节点集群==========

```
# 节点1 shard1-47017.conf
dbpath=/data/mongo/shard1
port=47017
fork=true
logpath=logs/shard1.log
shardsvr=true

# 节点2 shard2-47018.conf
dbpath=/data/mongo/shard2
port=47018
fork=true
logpath=logs/shard2.log
shardsvr=true
```

配置 路由节点 mongos ==========

节点 route-27017.conf

```
port=27017
bind_ip=0.0.0.0
fork=true
logpath=logs/route.log
configdb=configCluster/127.0.0.1:37017,127.0.0.1:37018
```

// 添加分片节点

```
sh.status()
sh.addShard("127.0.0.1:47017");
sh.addShard("127.0.0.1:47018");
```

为数据库开启分片功能

```
sh.enableSharding("tuling")
```

为指定集合开启分片功能

```
 sh.shardCollection("tuling.emp",{"_id":1})
```

修改分片大小

```
 use config
 db.settings.find()
db.settings.save({_id:"chunksize",value:1})
```

尝试插入1万条数据：

```
 for(var i=1;i<=100000;i++){
                db.emp.insert({"_id":i,"name":"copy"+i});
}
```

四、用户管理与数据集验证

// 创建管理员用户

```
use admin;
db.createUser({"user":"admin","pwd":"123456","roles":["root"]})
#验证用户信息
db.auth("admin","123456")
#查看用户信息
db.getUsers()
# 修改密码
db.changeUserPassword("admin","123456")
```

以auth 方式启动mongod，需要添加auth=true 参数， mongdb 的权限体系才会起作用：

```
#以auth 方向启动mongod （也可以在mongo.conf 中添加auth=true 参数）
./bin/mongod -f conf/mongo.conf --auth
# 验证用户
use admin;
db.auth("admin","123456")
```

创建只读用户

```
db.createUser({"user":"dev","pwd":"123456","roles":["read"]})
```

重新登陆 验证用户权限

```
use luban  ;
db.auth("dev","123456")
```