

NoSQL数据库管理

NSD NoSQL

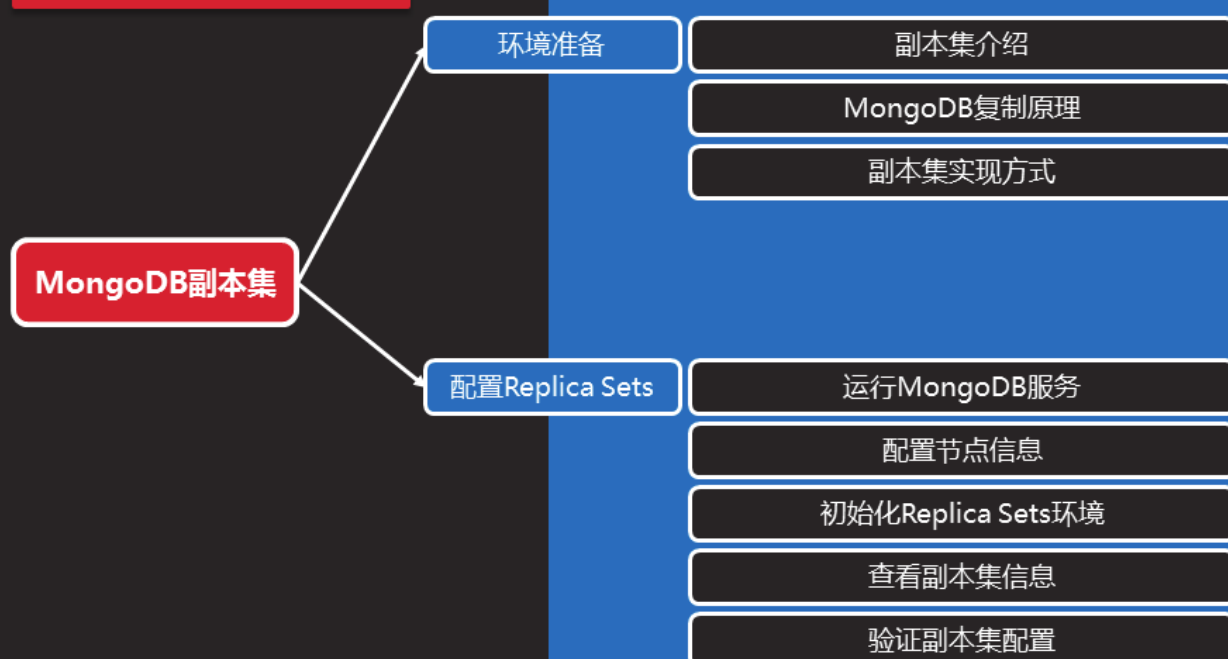
DAY05

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	MongoDB副本集
	10:30 ~ 11:20	
	11:30 ~ 12:00	
下午	14:00 ~ 14:50	文档管理
	15:00 ~ 15:50	
	16:10 ~ 17:00	
	17:10 ~ 18:00	总结和答疑



MongoDB副本集

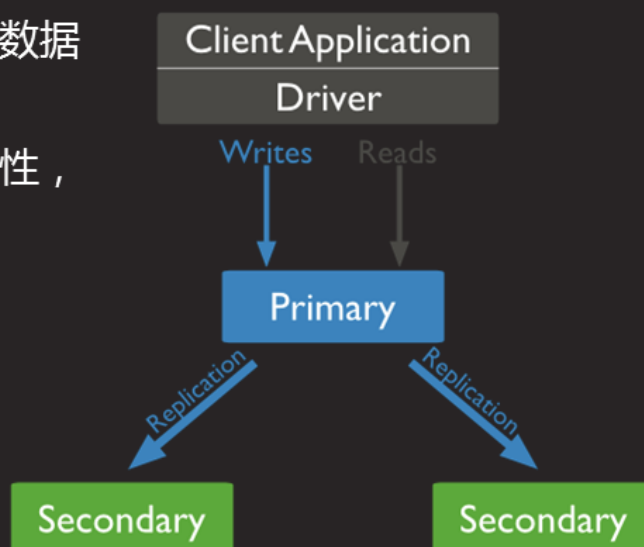


环境准备

副本集介绍

- 也称为MongoDB复制
 - 指在多个服务器上存储数据副本，并实现数据同步
 - 提高数据可用性、安全性，方便数据故障恢复

知识讲解



MongoDB复制原理

知识讲解

- 副本集工作过程
 - 至少需要两个节点。其中一个是主节点，负责处理客户端请求，其余是从节点，负责复制主节点数据
 - 常见搭配方式：一主一从、一主多从
 - 主节点记录所有操作oplog，从节点定期轮询主节点获取这些操作，然后对自己的数据副本执行这些操作，从而保证从节点的数据与主节点一致



副本集实现方式

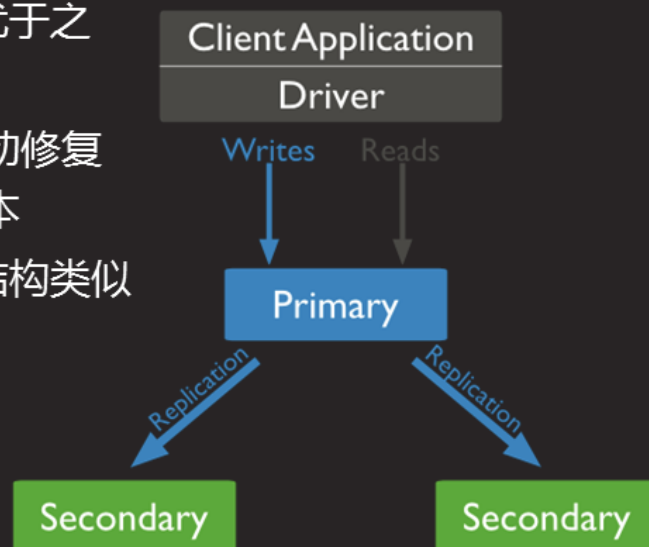
知识讲解

- Master-Slave 主从复制
 - 启动一台服务器时加上“-master”参数，作为主节点
 - 启动其他服务器时加上“-slave”和“-source”参数，作为从节点
- 主从复制的优点
 - 从节点可以提供数据查询，降低主节点的访问压力
 - 由从节点执行备份，避免锁定主节点数据
 - 当主节点故障时，可快速切换到从节点，实现高可用



副本集实现方式（续1）

- Replica Sets副本集
 - 从1.6 版本开始支持，优于之前的replication
 - 支持故障自动切换、自动修复成员节点，降低运维成本
 - Replica Sets副本集的结构类似高可用集群



知识讲解



配置Replica Sets

运行MongoDB服务

知识讲解

- 启动服务时，指定主机所在副本集名称
 - 所有副本集成员使用相同的副本集名称
 - `--replSet rs1` //指定副本集名称

```
[root@server0 ~]# mkdir -p /data/db  
[root@server0 ~]# ./mongod --bind_ip 192.168.4.61 \  
--logpath=/var/log/mongod.log --replSet rs1 &
```



配置节点信息

知识讲解

- 在任意一台主机连接mongod服务，执行如下操作

```
[root@server0 ~]# ./mongo --host 192.168.4.61  
config = {  
  _id:"rs1",  
  members:[  
    {_id:0,host:"IP地址:端口"},  
    {_id:1,host:"IP地址:端口"},  
    {_id:2,host:"IP地址:端口"}  
  ]  
};
```



初始化Replica Sets环境

- 执行如下命令
 - > `rs.initiate(config)`

知识讲解

```
>
> rs.initiate(config)
{
  "ok" : 1,
  "operationTime" : Timestamp(1526483504, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1526483504, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
rs1:SECONDARY> _
```



查看副本集信息

- 查看状态信息
 - > `rs.status()`
- 查看是否是master库
 - > `rs.isMaster()`

知识讲解



验证副本集配置

知识讲解

- 同步数据验证，允许从库查看数据
 - `>db.getMongo().setSlaveOk()`
- 自动切换主库验证
 - `> rs.isMaster()`



案例1：配置MongoDB副本集

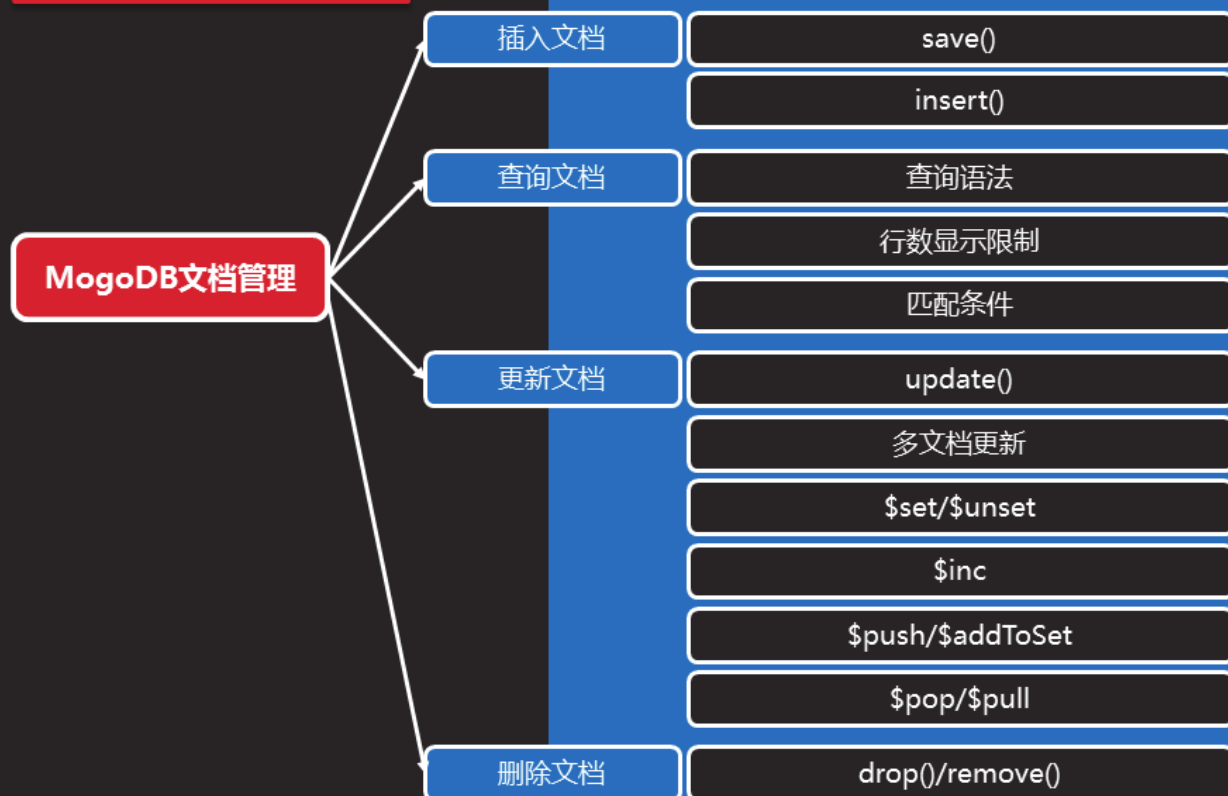
具体要求：

- 准备3台mongodb服务器
- 配置副本集服务
- 验证副本集配置

课堂练习



MongoDB文档管理



插入文档

save()

- 格式
 - > db.集合名.save({ key: "值" , key:" 值" })
- 注意
 - 集合不存在时创建集合，然后再插入记录
 - _id字段值已存在时，修改文档字段值
 - _id字段值不存在时，插入文档

知识讲解



insert()

- 格式
 - > db.集合名.insert({key:"值" , key:"值"})
- 注意
 - 集合不存在时创建集合，然后再插入记录
 - _id字段值已存在时，放弃插入
 - _id字段值不存在时，插入文档

知识讲解



insert()(续1)

- 插入多条记录

```
> db.集合名.insertMany(  
    [  
        {name:"xiaojiu",age:19} ,  
        {name:"laoshi",email:"yaya@tedu.cn"}  
    ]  
)
```

知识讲解



查询文档

查询语法

知识讲解

- 显示所有行（默认输出20行，输入it可显示后续行）

```
> db.集合名.find()
```

- 显示第1行

```
> db.集合名.findOne()
```

- 指定查询条件并指定显示的字段

```
> db.集合名.find( {条件},{定义显示的字段} )
```

```
> db.user.find({}, {_id:0,name:1,shell:1})
```

//0 不显示，1 显示



行数显示限制

知识讲解

- limit(数字) //显示前几行

```
> db.集合名.find().limit(3)
```

- skip(数字) //跳过前几行

```
> db.集合名.find().skip(2)
```

- sort(字段名) //1升序，-1降序

```
> db.集合名.find().sort(age:1|-1)
```

```
>
```

```
db.user.find({shell:"/sbin/nologin"}, {_id:0,name:1,uid:1,shell:1}).skip(2).limit(2)
```



匹配条件

知识讲解

- 简单条件

- > db.集合名.find({key:"值"})
- > db.集合名.find({key:"值" , keyname:"值"})
- > db.user.find({shell:"/bin/bash"})
- > db.user.find({shell:"/bin/bash",name:"root"})



匹配条件（续1）

知识讲解

- 范围比较

- \$in 在...里
- \$nin 不在...里
- \$or 或

- > db.user.find({uid:{\$in:[1,6,9]}})
- > db.user.find({uid:{\$nin:[1,6,9]}})
- > db.user.find({\$or: [{name:"root"},{uid:1}]})



匹配条件（续2）

知识讲解

- 正则匹配

```
> db.user.find({name: /^a/ })
```

- 数值比较

– \$lt \$lte \$gt \$gte \$ne

– < <= > >= !=

```
> db.user.find( { uid: { $gte:10,$lte:40} } , {_id:0,name:1,uid:1})
```

```
> db.user.find({uid:{$lte:5}})
```



匹配条件（续3）

知识讲解

- 匹配null,也可以匹配没有的字段

```
> db.user.save({name:null,uid:null})
```

```
> db.user.find({name:null})
```

```
> db.user.find({"_id":
```

```
ObjectId("5afd0ddb42772e7e458fc75"),"name" : null, "uid" :  
null })
```



更新文档

update()

- 语法格式

> db.集合名.update({条件},{修改的字段})

```
> db.user3.find({uid: {$lte: 3}}, {_id: 0})
```

```
> db.user3.update({uid: {$lte: 3}}, {password: "888"})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.user3.find({uid: {$lte: 3}}, {_id: 0})
```

```
> db.user3.find({password: "888"})
{ "_id" : ObjectId("5b0a5d95f72bcc99281cb118"), "password" : "888" }
```

把文件的其他字段都删除了，只留下了password字段，且只修改与条件匹配的第1行!!!

多文档更新

知识讲解

- 语法格式：默认只更新与条件匹配的第1行
 - > `db.user.update({条件},{set:{修改的字段}},false,true)`
 - > `db.user.update({name:"bin"},{set:{password:"abc12123"}},false,true)`



\$set/\$unset

知识讲解

- \$set 条件匹配时，修改指定字段的值
 - > `db.user.update({条件},{set: {修改的字段}})`
 - > `db.user3.update({name: "bin" },{set:{password: "A" }})`
- \$unset 删除与条件匹配文档的字段
 - > `db.集合名.update({条件},{unset:{key:values}})`
 - > `db.user3.update({name:"bin"},{unset:{password:"A"}})`



\$inc

知识讲解

- \$inc 条件匹配时，字段值自加或自减

> db.集合名.update({条件},{ \$inc:{字段名:数字}})

> db.user.update({name:"bin"},{ \$inc:{uid:2}}) //字段值自加2

> db.user.update({name: "bin" },{ \$inc:{uid:-1}}) //字段值自减1

+num 自增，-num 自减



\$push/\$addToSet

知识讲解

- \$push 向数组中添加新元素

> db.集合名.update({条件},{ \$push:{数组名:"值"}})

> db.user.insert({name:"bob",likes:["a","b","c","d","e","f"]})

> db.user.update({name: "bob" },{ \$push:{likes: "w"}})

- \$addToSet 避免重复添加

> db.集合名.update({条件},{ \$addToSet:{数组名:"值"}})

> db.user.update({name:"bob"},{ \$addToSet:{likes:"f"}})



\$pop/\$pull

知识讲解

- \$pop 从数组头部删除一个元素

> db.集合名.update({条件},{ \$pop:{数组名:数字}})

> db.user.update({name:"bob"},{ \$pop:{likes:1}})

> db.user.update({name:"bob"},{ \$pop:{likes:-1}})

- \$pull 删除数组指定元素

> db.集合名.update({条件},{ \$pull:{数组名:值}})

> db.user.update({name:"bob"},{ \$pull:{likes:"b"}})

1 删除数组尾部元素
-1 删除数组头部元素



删除文档

\$drop/\$remove

知识讲解

- \$drop 删除集合的同时删除索引
 - > db.集合名.drop()
 - > db.user.drop()
- \$remove() 删除文档时不删除索引
 - > db.集合名.remove({}) //删除所有文档
 - > db.集合名.remove({条件}) //删除与条件匹配的文档
 - > db.user.remove({uid:{\$lte:10}})
 - > db.user.remove({})



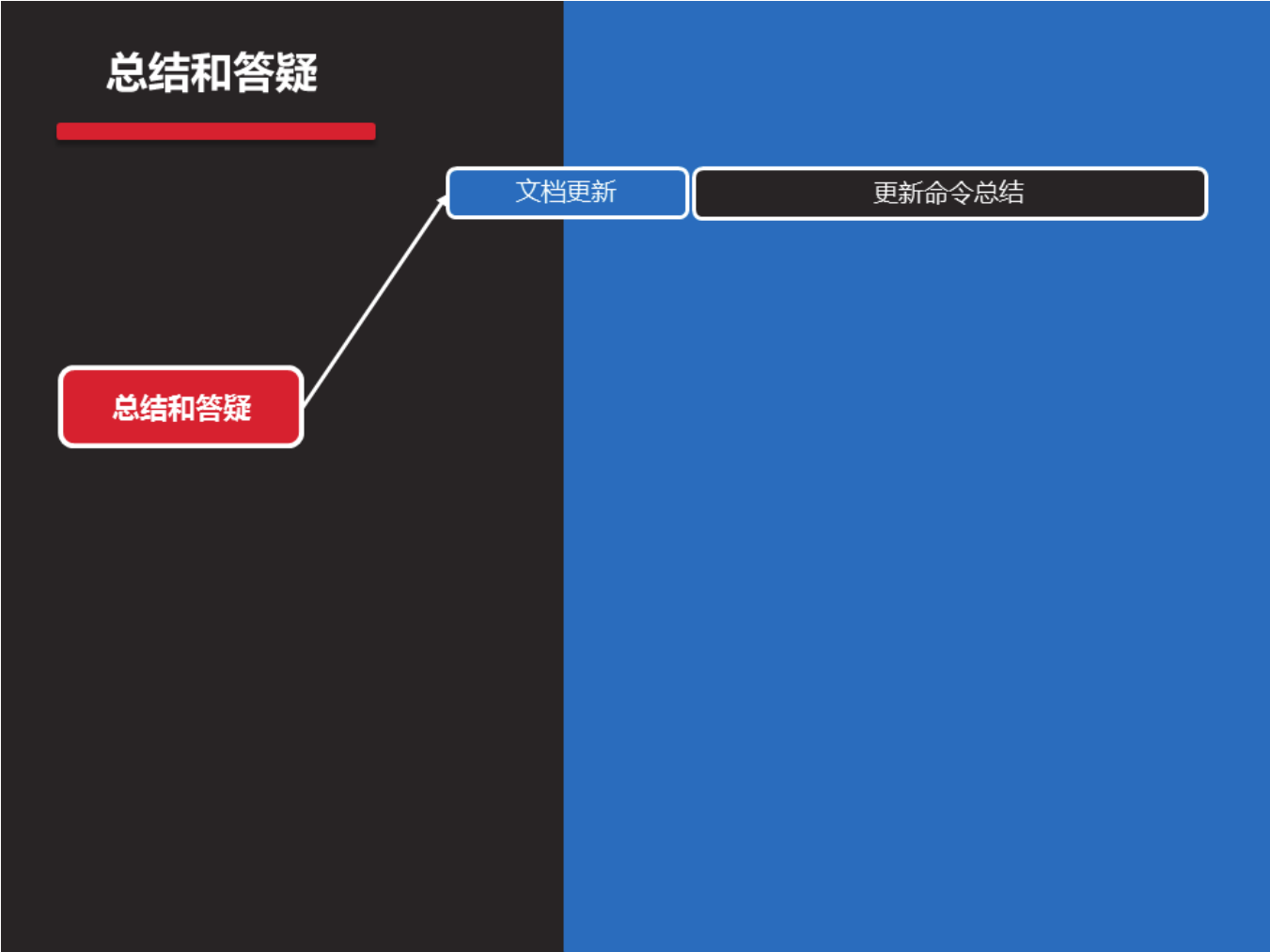
案例2：文档管理


基于MongoDB环境完成下列练习：

- 插入文档
- 查询文档
- 更新文档
- 删除文档

课堂练习








达内教育

文档更新



更新命令总结

知识讲解

类 型	用 途
\$set	修改文档指定字段的值
\$unset	删除记录中的字段
\$push	向数组内添加新元素
\$pull	删除数组中的指定元素
\$pop	删除数组头尾部元素
\$addToSet	避免数组重复赋值
\$inc	字段自加或自减

