私塾在线《高级软件架构师实战培训阶段一》

——跟着cc学架构系列精品教程

本部分课程概览

- 根据实际的应用需要，学习要用到的MongoDB的知识，以快速上手、理解并掌握MongoDB
- 一：MongoDB简介
  包括：MongoDB是什么、能干什么、特点、NoSQL简介、为什么需要NoSQL、CAP原理、BASE原则、NoSql优缺点等
- 二：MongoDB安装和基本使用
  包括：安装和基本使用、启动参数说明、MongoDB基本概念、基本数据类型等
- 三：MongoDB增删改操作
  包括：多种操作命令、多种修改器的使用
- 四：MongoDB查询操作
  包括：指定要返回的键、按条件查询、多种比较符和比较命令、数组查询、内嵌文档查询、$where查询、聚合命令、分页查询、游标、存储过程等
- 五：聚合框架
  包括：概念、使用方式、各种管道操作符的应用、MapReduce的开发和应用等

本部分课程概览

- 六：理解MongoDB的文档存储
  包括：理解MongoDB的文档存储的机制和原理，并示范优化文档增长的方法
- 七：MongoDB的索引
  包括：对索引的各种操作命令、索引的使用分析、索引的类型等
- 八：Capped集合和GridFS
  包括：Capped集合的概念和使用、GridFS的使用
- 九：MongoDB的副本集
  包括：副本集的创建、初始化、使用维护、和多种成员配置选项等
- 十：副本集的基本原理
  包括：理解Oplog、理解复制的过程、理解初始化同步、处理陈旧数据、理解心跳、理解回滚机制等
- 十一：副本集的管理
  包括：单机模式启动成员，维护副本集，强制重新配置，把主节点变为备份节点，阻止选举，不作为复制源、MongoDB的主从复制等

本部分课程概览

- 十二：MongoDB的分片
  包括：理解MongoDB的分片、各部分的组成和功能、分片的具体做法、理解分片的原理、块、数据拆分、均衡器、限制分片大小、理解数据分配方式等
- 十三：MongoDB分片片键的选择
  包括：重要性、基本规则、常见片键类型分析、好的片键建议等
- 十四：分片的管理
  包括：查看很多关于分片的信息，检查配置，添加和删除分片等
- 十五：杂项技术
  包括：监控应用状态、用户身份验证、备份和恢复、数据导入导出等
- 十六：Java操作MongoDB
  包括：获取驱动，构建环境，CRUD实现等
- 十七：MondoDB和Spring集成开发
  包括：构建环境、Spring配置、MongoTemplate开发使用等
- 十八： MongoDB应用建议及最佳实践

MongoDB简介-1

- MongoDB是什么

  MongoDB是一个使用C++编写的、开源的、面向文档的NoSQL(Not Only SQL)数据库，也是当前最热门的NoSql数据库之一。

- NoSQL简介

  NoSQL的意思是“不仅仅是SQL”，是目前流行的“非关系型数据库”的统称。

  常见的NoSQL数据库如：Redis、CouchDB、MongoDB、HBase、Cassandra等

- 为什么需要NoSQL？

  简单的说，就是为了解决在web2.0时代，出现的三高要求：

1：对数据库高并发读写的需求

2：对海量数据的高效率存储和访问的需求

3：对数据库的高可扩展性和高可用性的需求

而RDB里面的一些特性，在web2.0里面往往变得不那么重要，比如：

1：数据库事务一致性

2：数据库的实时读写

3：复杂的SQL查询，特别是多表关联查询

- tbl_user
- Uuid userId name
- 1 u1 zhangsan
- 2 u2

- bson
- {“uuid”:1,”userId”:”u1”,”name”:zhangsan}
- {“uuid”:2,”userId”:”u2”,age:13}

MongoDB简介-2

- CAP定理，又被称作布鲁尔定理（Eric Brewer）

  它指出对于一个分布式计算系统来说，不可能同时满足以下三点:

1：强一致性(Consistency)：系统在执行过某项操作后仍然处于一致的，在分布式系统中，更新操作执行成功后所有的用户都应该读取到最新的值，这样的系统被认为具有强一致性

2：可用性(Availability)：每一个操作总是能够在一定的时间内返回结果

3：分区容错性(Partition tolerance)：系统在存在网络分区的情况下仍然可以接受请求并处理，这里网络分区是指由于某种原因网络被分成若干个孤立区域，而区域之间互不相通

- 根据CAP原理将数据库分成了满足CA原则、满足CP原则和满足AP原则三大类：

1：CA：单点集群，满足一致性，可用性，通常在可扩展性上不太强大，比如RDB

2：CP：满足一致性和分区容错性，通常性能不是特别高，如分布式数据库

3：AP：满足可用性和分区容错性，通常可能对一致性要求低一些，如大多数的NoSQL

- BASE（Basically Available，Soft-state，Eventual consistency ）

1：基本可用（Basically Available）：系统能够基本运行、一直提供服务。

2：软状态（Soft-state）：系统不要求一直保持强一致状态。

3：最终一致性（Eventual consistency）：系统需要在某一时刻后达到一致性要求

MongoDB简介-3

- NoSQL的优点
  - 扩展简单方便，尤其是水平横向扩展
    （纵向扩展是指用更强的机器；横向扩展是指把数据分散到多个机器）
  - 读写快速高效，多数都会映射到内存操作
  - 成本低廉，用普通机器，分布式集群即可
  - 数据模型灵活，没有固定的数据模型
- NoSQL的缺点
  - 不提供对SQL的支持
  - 现有产品还不够成熟稳定，功能也还有待加强
- MongoDB特点
  - 高性能、易于使用、易于扩展、功能丰富
  - 面向集合存储，模式自由
  - 支持动态查询，支持javascript表达式查询
  - 支持索引
  - 支持副本集复制和自动故障恢复
  - 自动处理分片
  - 支持二进制和大型对象数据
  - 文件存储格式为BSON（JSON的一种扩展）

MongoDB安装和基本使用

- 安装

  1：去官方下载最新的包，http://www.mongodb.org/downloads

  2：然后tar zvxf 解压

  3：拷贝到相应的文件夹即可

- 在Shell里面启动

  1：在MongoDB的文件夹下创建dbs和logs的文件夹

  2：到bin下，运行 ./mongod --dbpath ../dbs，就可以启动数据库了

  当然，也可以通过 --logpath 来指定日志文件的路径，需要指定到文件。

  3：可以把启动的参数设置到一个配置文件中，然后在启动的时候通过-f进行指定

  4：MongoDB默认会监听27017端口，可以通过 --port来指定主端口

  5：可以通过 ./mongod --help来查看启动时可以指定的参数

- 在后台启动

  使用--fork选项，将会通知mongodb在后台运行。也可以配置到文件里面去，设置fork=true即可

- 关闭

  1：如果是在Shell里面启动的，ctrl+c退出shell就关闭了

  2：如果是在后台启动的，运行 pkill mongod

  3：也可以进入javascript shell，切换到admin数据库，运行db.shutdownServer()

MongoDB启动参数说明-1

n 基本配置

--quiet # 安静输出
--port arg # 指定服务端口号，默认端口27017
--bind_ip arg # 绑定服务IP，若绑定127.0.0.1，则只能本机访问，不指定默认本地所有IP
--logpath arg # 指定MongoDB日志文件，注意是指定文件不是目录
--logappend # 使用追加的方式写日志
--pidfilepath arg # PID File 的完整路径，如果没有设置，则没有PID文件
--keyFile arg # 集群的私钥的完整路径，只对于Replica Set 架构有效
--unixSocketPrefix arg # UNIX域套接字替代目录,(默认为 /tmp)
--fork # 以守护进程的方式运行MongoDB，创建服务器进程
--auth # 启用验证
--cpu # 定期显示CPU的CPU利用率和iowait
--dbpath arg # 指定数据库路径
--diaglog arg # diaglog选项 0=off 1=W 2=R 3=both 7=W+some reads
--directoryperdb # 设置每个数据库将被保存在一个单独的目录
--journal # 启用日志选项，MongoDB的数据操作将会写入到journal文件夹的文件里
--journalOptions arg # 启用日志诊断选项
--ipv6 # 启用IPv6选项
--jsonp # 允许JSONP形式通过HTTP访问（有安全影响）
--maxConns arg # 最大同时连接数 默认2000
--noauth # 不启用验证
--nohttpinterface # 关闭http接口，默认关闭27018端口访问

MongoDB启动参数说明-2

--nopreallocc # 禁用数据文件预分配(往往影响性能)
--noscripting # 禁用脚本引擎
--notablescan # 不允许表扫描
--nounixsocket # 禁用Unix套接字监听
--nssize arg (=16) # 设置信数据库.ns文件大小(MB)
--objcheck # 在收到客户数据,检查的有效性，
--profile arg # 档案参数 0=off 1=slow, 2=all
--quota # 限制每个数据库的文件数，设置默认为8
--quotaFiles arg # number of files allower per db, requires --quota
--rest # 开启简单的rest API
--repair # 修复所有数据库run repair on all dbs
--repairpath arg # 修复库生成的文件的目录,默认为目录名称dbpath
--slowms arg (=100) # value of slow for profile and console log
--smallfiles # 使用较小的默认文件
--syncdelay arg (=60) # 数据写入磁盘的时间秒数(0=never,不推荐)
--sysinfo # 打印一些诊断系统信息
--upgrade # 如果需要升级数据库

- Replicaton 参数

--fastsync # 从一个dbpath里启用从库复制服务，该dbpath的数据库是主库的快照，可用于快速启用同步
--autoresync # 如果从库与主库同步数据差得多，自动重新同步，
--oplogSize arg # 设置oplog的大小(MB)

MongoDB启动参数说明-3

- 主/从参数

--master # 主库模式
--slave # 从库模式
--source arg # 从库 端口号
--only arg # 指定单一的数据库复制
--slavedelay arg # 设置从库同步主库的延迟时间

- Replica set(副本集)选项：

--replSet arg # 设置副本集名称

- Sharding(分片)选项

--configsvr # 声明这是一个集群的config服务,默认端口27019，默认目录 /data/configdb
--shardsvr # 声明这是一个集群的分片,默认端口27018
--noMoveParanoia # 关闭偏执为moveChunk数据保存

MongoDB基本概念-1

- 数据库

  MongoDB的一个实例可以拥有一个或多个相互独立的数据库，每个数据库都有自己的集合
- 集合

  集合可以看作是拥有动态模式的表
- 文档

  文档是MongoDB中基本的数据单元，类似于RDB的行。

  文档是键值对的一个有序集合。在JS中，文档被表示成对象。
- _id

  每个文档都有个特殊的“_id”，在文档所属集合中是唯一的
- JavaScript shell

  MongoDB自带了一个功能强大的JavaScript Shell，可以用于管理或操作MongoDB
- MongoDB和RDB的一些对比

1：都有数据库的概念
2：集合 --〉RDB的表
3：文档 --〉RDB表中的一条记录
4：文档对象里面的 key --〉 RDB表中的字段
5：文档对象里面的 value--〉 RDB表中字段的值
6：MongoDB中没有主外键的概念

MongoDB基本概念-2

- 数据库名称定义规则

1：不能是空串
2：不得含有/、\、？、$、空格、空字符等等，基本只能使用ASCII中的字母和数字
3：区分大小写，建议全部小写
4：最多为64字节
5：不得使用保留的数据库名，比如：admin，local，config

注意：数据库最终会成为文件，数据库名就是文件的名称

- 集合名称定义规则

1：不能是空串
2：不能包含\0字符（空字符），这个字符表示集合名的结束，也不能包含”$”
3：不能以”system.”开头，这是为系统集合保留的前缀

- 文档的键的定义规则

1：不能包含\0字符（空字符），这个字符表示键的结束
2：“.”和“$”是被保留的，只能在特定环境下用
3：区分类型，同时也区分大小写
4：键不能重复

注意：文档的键值对是有顺序的，相同的键值对如果有不同顺序的话，也是不同的文档

MongoDB基本概念-3

n MongoDB基本的数据类型

数据类型	描述	举例
null	表示空值或者未定义的对象	{"x":null}
布尔值	真或者假：true或者false	{"x":true}
32位整数	shell不支持该类型，默认会转换成64位浮点数，也可以使用NumberInt类，比如：	{ “x” :NumberInt(“3”)}
64位整数	shell不支持该类型，默认会转换成64位浮点数，也可以使用NumberLong类，比如：	{ “x” :NumberLong(“3”)}
64位浮点数	shell中的数字就是这一种类型	{"x"：3.14，"y"：3}
字符串	UTF-8字符串	{"foo":"bar"}
符号	shell不支持，shell会将数据库中的符号类型的数据自动转换成字符串	
对象id	文档的12字节的唯一id	{"id": ObjectId()}
日期	从标准纪元开始的毫秒数	{"date":new Date()}
正则表达式	文档中可以包含正则表达式，遵循JavaScript的语法	{"foo":/foobar/i}
代码	文档中可以包含JavaScript代码	{"x"：function() {}}
未定义	undefined	{"x"：undefined}
数组	值的集合或者列表	{"arr": ["a","b"]}
内嵌文档	文档可以作为文档中某个key的value	{"x":{"foo":"bar"}}

MongoDB增删改操作-1

- 运行shell，命令：mongo ip:port
- 显示现有的数据库，命令：show dbs 或者 databases;
- 显示当前使用的数据库，命令：db
- 切换当前使用的数据库，命令：use 数据库名称
- 创建数据库：MongoDB没有专门创建数据库的语句，可以使用“use” 来使用某个数据库，如果要使用的数据库不存在，那么将会创建一个，会在真正向该库加入文档后，保存成为文件。
- 删除数据库，命令：db.dropDatabase()
- 显示现有的集合，命令：show collections 或者 tables;
- 创建集合：在MongoDB中不用创建集合，因为没有固定的结构，直接使用db.集合名称.命令 来操作就可以了。如果非要显示创建集合的话，用：db.createCollecion(“集合名称”);
- 插入并保存文档

  insert方法，可以单独插入一个文档，也可以插入多个，用“[ ]”即可。注意：

1：MongoDB会为每个没有“_id”字段的文档自动添加一个”_id”字段

2：每个Doc必须小于16MB

3：可以在shell中执行Object.bsonsize(文档名称);来查看size大小

- 删除文档，命令：remove ， 可以按条件来删除

  只是删除文档，集合还在，如果使用 drop命令，会连带集合和索引都删掉
- 查看集合的状态信息：db.集合名.stats();

MongoDB增删改操作-2

- 查看集合中所有的文档，命令：db.集合名称.find();
- 查看集合中第一个文档，命令：db.集合名称.findOne({条件对象});
- 文档替换，命令： db.集合名称. update(条件，新的文档);
- 更新修改器，用来做复杂的更新操作

1：$set ：指定一个字段的值，如果字段不存在，会创建一个
2：$unset ：删掉某个字段
3：$inc ：用来增加已有键的值，如果字段不存在，会创建一个。只能用于整型、长整型、或双精度浮点型的值。
4：$push：向已有数组的末尾加入一个元素，要是没有就新建一个数组
5：$each：通过一次$push来操作多个值
6：$slice：限制数组只包含最后加入的n个元素，其值必须是负整数
7：$sort：对数组中的元素，按照指定的字段来对数据进行排序（1为升序，-1为降序），然后再按照slice删除。注意：不能只将$slice或者$sort与$push配合使用，且必须使用$each
8：$ne：判断一个值是否在数组中，如果不在则添加进去
9：$addToSet：将数组作为数据集使用，以保证数组内的元素不会重复
10：$pop ：从数组一端删除元素，{$pop:{key:1}}，从末尾删掉一个，-1则从头部删除
11：$pull：按照条件来删除所有匹配的元素
12：$：用来修改第一个匹配的元素

MongoDB增删改操作-3

- save方法

  如果文档存在就更新，不存在就新建，主要根据”_id”来判断。

- upsert

  找到了符合条件的文档就更新，否则会以这个条件和更新文档来创建一个新文档。

  指定update方法的第三个参数为true，可表示是upsert

- 更新多个文档

  MongoDB默认只更新符合条件的第一个文档，要更新所有匹配的文档，把第4个参数设置为true。注意：

1：只能用在$XXX的操作中

2：最好每次都显示的指定update的第4个参数，以防止服务器使用默认行为

- 查询更新了多少个文档

  使用命令：getLastError ，返回最后一次操作的相关信息，里面的n就是更新的文档的数量。形如：db.runCommand({"getLastError":1});

MongoDB查询操作-1

- 指定需要返回的键

  在find方法的第二个参数进行指定。默认情况下，始终会返回”_id”，可以通过设置字段为0来表示不返回这个字段。

- 按条件查询

  在find方法里面加入条件数据即可，find方法的第一个参数就是。

  注意：条件数据必须是常量值，不能是另外的字段的数据

1：比较操作：$lt,$lte,$gt,$gte,$ne

2：$and：包含多个条件，他们之间为and的关系

3：$or ：包含多个条件，他们之间为or的关系 ，$nor相当于or取反

4：$not：用作其他条件之上，取反

5：$mod：将查询的值除以第一个给定的值，如果余数等于等二个值则匹配成功

6：$in ：查询一个键的多个值，只要键匹配其中一个即可 ，$nin为不包含

7：$all：键需要匹配所有的值

8：$exists：检查某个键是否存在，1表示存在，0表示不存在

9：null类型：不仅能匹配键的值为null，还匹配键不存在的情况

MongoDB查询操作-2

- 正则表达式

  MongoDB使用Perl兼容的正则表达式（PCRE），比如：

  db.users.find({“name”:/sishuok/i}); 又比如：

  db.users.find({“name”:/^sishuok/});

- 查询数组

1：单个元素匹配，就跟前面写条件一样，{key：value}

2：多个元素匹配，使用$all，{key：{$all:[a,b]}}，元素的顺序无所谓

3：可以使用索引指定查询数组特定位置， {“key.索引号”：value}

4：查询某个长度的数组，使用$size

5：指定子集，使用$slice，正数是前面多少条，负数是尾部多少条，也可以指定偏移量和要返回的元素数量，比如：$slice:[50,10]

6：可以使用$来指定符合条件的任意一个数组元素，如：{”users.$”:1}

7：$elemMatch：要求同时使用多个条件语句来对一个数组元素进行比较判断

MongoDB查询操作-3

- 查询内嵌文档

1：查询整个内嵌文档与普通查询是一样的

2：如果要指定键值匹配，可以使用“.” 操作符，比如：{“name.first”:”a” ,
“name.last”:”b”}

3：如果要正确的指定一组条件，那就需要使用$elemMatch，以实现对内嵌文档的多个键进行匹配操作

注意：内嵌文档的查询必须要整个文档完全匹配

- $where查询

在查询中执行任意的JavaScript，通过编程来解决查询的匹配问题，方法返回boolean值。

```
function t1(){
    for(var a in this){
          if(a=="a"){return true;}
    }
    return false;
}
```

使用的时候：db.users.find({"$where":t1});

注意：$where性能较差，安全性也是问题，所以不到万不得已，不要使用

MongoDB查询操作-4

- 查询记录条数的命令：count

1：直接使用count()的话，得到的是整个记录的条数

2：如果要获取按条件查询后记录的条数，需要指定count(true或者非0的数)

- 限制返回的记录条数的命令：limit(要返回的条数)
- 限制返回的记录条数起点的命令：skip(从第几条开始返回)
- 排序的命令：sort({要排序的字段：1为升序，-1为降序})

1：可以对多个字段进行排序

2：MongoDB处理不同类型的数据是有一定顺序的，有时候一个键有多种类型的值，其排序顺序是预先定义好的，从小到大如下：

（1）最小值　（2）null　（3）数字　（4）字符串
（5）对象/文档　（6）数组　（7）二进制数据　（8）对象id
（9）布尔类型　（10）日期型　（11）时间戳　（12）正则表达式
（13）最大值

- 分页查询：组合使用limit,skipt和sort

  当然也可以使用其他方式来分页，比如采用自定义的id，然后根据id来分页
- 查询给定键的所有不重复的数据，命令：distinct

  语法：db.runCommand({“distinct”:集合名,“key”:”获得不重复数据的字段”});

MongoDB查询操作-5

- 游标

1：获取游标，示例如下：

```
var c = db.users.find();
```

2：循环游标，可以用集合的方式，示例如下：

```
while(c.hasNext()){
        printjson(c.next());
}
```

3：也可以使用forEach来循环，示例如下：

```
c.forEach(function(obj){
        print(obj);
});
```

- 存储过程

1：MongoDB的存储过程其实就是个自定义的js函数

2：使用db.system.js.save({“_id”:名称,value:函数});

3：可以通过如下命令查看：db.system.js.find();

4：可以通过如下命令调用：db.eval(名称);

聚合框架-1

- 简介

  MongoDB的聚合框架，主要用来对集合中的文档进行变换和组合，从而对数据进行分析以加以利用。

  聚合框架的基本思路是：采用多个构件来创建一个管道，用于对一连串的文档进行处理。这些构件包括：筛选(filtering)、投影(projecting)、分组(grouping)、排序(sorting)、限制(limiting)和跳过(skipping)。

- 使用聚合框架的方式

  db.集合.aggregate（构件1，构件2…）

  注意：由于聚合的结果要返回到客户端，因此聚合结果必须限制在16M以内，这是MongoDB支持的最大响应消息的大小。

- 准备样例数据

```
for(var i=0;i<100;i++){
    for(var j=0;j<4;j++){
        db.scores.insert({"studentId":"s"+i,"course":"课程"+j,"score":Math.random()*100});
    }
}
```

- 示例要完成的功能

  找出考80分以上的课程门数最多的3个学生

聚合框架-2

- 是用聚合框架来完成功能的步骤

1：找到所有考了80分以上的学生，不区分课程

```
{"$match":{"score":{$gte:80}}}
```

2：将每个学生的名字投影出来

```
{"$project":{"studentId":1}}
```

3：对学生的名字排序，某个学生的名字出现一次，就给他加1

```
{"$group":{"_id":"$studentId","count":{"$sum":1}}}
```

4：对结果集按照count进行降序排列

```
{"$sort":{"count":-1}}
```

5：返回前面的3条数据

```
{"$limit":3}
```

最终执行的语句就是：

```
db.scores.aggregate(
  {"$match":{"score":{$gte:80}}}
, {"$project":{"studentId":1}}
, {"$group":{"_id":"$studentId","count":{"$sum":1}}}
, {"$sort":{"count":-1}}
, {"$limit":3}
)
```

聚合框架-3

- 管道操作符简介

每个操作符接受一系列的文档，对这些文档做相应的处理，然后把转换后的文档作为结果传递给下一个操作符。最后一个操作符会将结果返回。

不同的管道操作符，可以按照任意顺序，任意个数组合在一起使用。

- 管道操作符$match

用于对文档集合进行筛选，里面可以使用所有常规的查询操作符。通常会放置在管道最前面的位置，理由如下：

1：快速将不需要的文档过滤，减少后续操作的数据量

2：在投影和分组之前做筛选，查询可以使用索引

- 管道操作符$project

用来从文档中提取字段，可以指定包含和排除字段，也可以重命名字段。比如要将studentId改为sid，如下：

```
db.scores.aggregate({"$project":{"sid":"$studentId"}})
```

管道操作符还可以使用表达式，以满足更复杂的需求。

聚合框架-4

- 管道操作符$project的数学表达式

  比如给成绩集体加20分，如下：

  db.scores.aggregate({"$project":{"newScore":{$add:["$score",20]}}})

支持的操作符和相应语法：

1：$add : [expr1[,expr2,…exprn]]

2：$subtract:[expr1,expr2]

3：$multiply:[expr1[,expr2,…exprn]]

4：$divice:[expr1,expr2]

5：$mod:[expr1,expr2]

- 管道操作符$project的日期表达式

  聚合框架包含了一些用于提取日期信息的表达式，如下：

  $year、$month、$week、$dayOfMonth、$dayOfWeek、$dayOfYear、$hour、$minute、$second 。

注意：这些只能操作日期型的字段，不能操作数据，使用示例：

{"$project":{"opeDay":{"$dayOfMonth":"$recoredTime"}}}

聚合框架-5

- 管道操作符$project的字符串表达式

1：$substr ： [expr,开始位置,要取的字节个数]

2：$concat:[expr1[,expr2,…exprn]]

3：$toLower:expr

4：$toUpper:expr

例如：{"$project":{"sid":{$concat:["$studentId","cc"]}}}

- 管道操作符$project的逻辑表达式

1：$cmp:[expr1,expr2] ：比较两个表达式，0表示相等，正数前面的大，负数后面的大

2：$strcasecmp:[string1,string2] ：比较两个字符串，区分大小写，只对由罗马字符组成的字符串有效

3：$eq、$ne、$gt、$gte、$lt、$lte ： [expr1,expr2]

4：$and、$or、$not

5：$cond:[booleanExpr,trueExpr,falseExpr]:如果boolean表达式为true，返回true表达式，否则返回false表达式

6：$ifNull:[expr,otherExpr]:如果expr为null，返回otherExpr，否则返回expr

例如：db.scores.aggregate({"$project":{"newScore":{$cmp:["$studentId","sss"]}}})

聚合框架-6

- $group

用来将文档依据特定字段的不同值进行分组。选定了分组字段过后，就可以把这些字段传递给$group函数的“_id”字段了。例如：

```
db.scores.aggregate({ “$group” :{ “_id” : “$studentId” }});  或者是
db.scores.aggregate({"$group":{"_id":{"sid":"$studentId","score":"$score"}}});
```

- $group支持的操作符

1：$sum:value ： 对于每个文档，将value与计算结果相加
2：$avg:value ： 返回每个分组的平均值
3：$max:expr ： 返回分组内的最大值
4：$min:expr ： 返回分组内的最小值
5：$first:expr ： 返回分组的第一个值，忽略其他的值，一般只有排序后，明确知道数据顺序的时候，这个操作才有意义
6：$last:expr ： 与上面一个相反，返回分组的最后一个值
7：$addToSet:expr ： 如果当前数组中不包含expr，那就将它加入到数组中
8：$push:expr：把expr加入到数组中

聚合框架-7

- 拆分命令：$unwind

  用来把数组中的每个值拆分成为单独的文档。

- 排序命令：$sort

  可以根据任何字段进行排序，与普通查询中的语法相同。如果要对大量的文档进行排序，强烈建议在管道的第一个阶段进行排序，这时可以使用索引。

- 常见聚合函数

1：count：用于返回集合中文档的数量

2：distinct：找出给定键的所有不同值，使用时必须指定集合和键，例如：

```
db.runCommand({"distinct":"users","key":"userId"});
```

- MapReduce介绍

  在MongoDB的聚合框架中，还可以使用MapReduce，它非常强大和灵活，但具有一定的复杂性，专门用于实现一些复杂的聚合功能。

  MongoDB中的MapReduce使用JavaScript来作为查询语言，因此能表达任意的逻辑，但是它运行非常慢，不应该用在实时的数据分析中。

聚合框架-8

- MapReduce的HelloWorld

实现的功能，找出集合中所有的键，并统计每个键出现的次数。

1：Map函数使用emit函数来返回要处理的值，示例如下：

```
var map = function(){
    for(var key in this){
        emit(key,{count:1});
    }
}
```

this表示对当前文档的引用。

2：reduce函数需要处理Map阶段或者是前一个reduce的数据，因此reduce返回的文档必须要能作为reduce的第二个参数的一个元素，示例如下：

```
var reduce = function(key,emits){
    var total = 0;
    for(var i in emits){
        total += emits[i].count;
    }
    return {"count":total};
};
```

聚合框架-9

3：运行MapReduce，示例如下：

```
var mr =
db.runCommand({"mapreduce":"users","map":map,"reduce":reduce,"out":"mrout"});
```

4：查询最终的结果，示例如下：

```
db.mrout.find();
```

- 还可以改变一下，比如统计userId中值，以及每个值出现的次数，就可以如下操作

1：修改map函数，示例如下：

```
var map = function(){
    emit(this.userId,{count:1});
};
```

2：reduce函数不用改

3：重新执行

```
db.runCommand({"mapreduce":"users","map":map,"reduce":reduce,"out":"mrout"});
```

4：查看最终结果：

```
db.mrout.find();
```

聚合框架-10

- 更多MapReduce可选的键

1：finalize:function ： 可以将reduce的结果发送到finalize，这是整个处理的最后一步

2：keeptemp:boolean ： 是否在连接关闭的时候，保存临时结果集合

3：query:document ： 在发送给map前对文档进行过滤

4：sort:document ： 在发送给map前对文档进行排序

5：limit:integer ： 发往map函数的文档数量上限

6：scope:document ： 可以在javascript中使用的变量

7：verbose:boolean ： 是否记录详细的服务器日志

示例：

```
var query = {"userId":{"$gt":"u2"}}
var sort = {"userId":1};
var finalize = function(key,value){
    return {"mykey":key,"myV":value};
};
var mr =
    db.runCommand({"mapreduce":"users","map":map,"reduce":reduce,"out":"mrout","que
    ry":query,"sort":sort,"limit":2,"finalize":finalize});
```

聚合框架-11

- 聚合命令group

用来对集合进行分组，分组过后，再对每一个分组内的文档进行聚合。

比如要对studentId进行分组，找到每个学生最高的分数，可以如下步骤进行：

1：测试数据就用聚合框架一开始准备的数据

2：使用group，示例如下：

```
db.runCommand({"group":{
    "ns":"scores",
    "key":{"studentId":1},
    "initial":{"score":0},
    "$reduce":function(doc,prev){
          if(doc.score > prev.score){
                prev.score = doc.score;
          }
    }
}});
```

ns：指定要分组的集合

key：指定分组的键

initial：每一组的reduce函数调用的时候，在开头的时候调用一次，以做初始化

$reduce：在每组中的每个文档上执行，系统会自动传入两个参数，doc是当前处理的文档，prev是本组前一次执行的结果文档

聚合框架-12

3：你还可以在group的时候添加条件，就是加入condition，示例如：

```
"condition":{"studentId":{"$lt":"s2"}}
```

4：同样可以使用finalizer来对reduce的结果进行最后的处理，比如要求每个学生的平均分，就可以先按照studentId分组，求出一个总的分数来，然后在finalizer里面，求平均分：

```
db.runCommand({"group":{
    "ns":"scores",
    "key":{"studentId":1},
    "initial":{"total":0},
    "$reduce":function(doc,prev){
          prev.total += doc.score;
    },
    "condition":{"studentId":{"$lt":"s2"}},
    "finalize":function(prev){
          prev.avg = prev.total/3;
    }
}});
```

注意：finalize是只在每组结果返回给用户前调用一次，也就是每组结果只调用一次

聚合框架-13

5：对于分组的key较为复杂的时候，还可以采用函数来做为键，比如让键不区分大小下，就可以如下定义：

```
db.runCommand({"group":{
    "ns":"scores",
    $keyf:function(doc){
          return {studentId:doc.studentId.toLowerCase()};
    },
    "initial":{"total":0},
    "$reduce":function(doc,prev){
          prev.total += doc.score;
    },
    "condition":{"$or":[{"studentId":{"$lt":"s2"}},{"studentId":"S0"}]},
    "finalize":function(prev){
          prev.avg = prev.total/3;
    }
}});
```

注意：要使用$keyf来定义函数作为键，另外一定要返回对象的格式

理解MongoDB的文档存储

- 将文档插入到MongoDB的时候，文档是按照插入的顺序，依次在磁盘上相邻保存

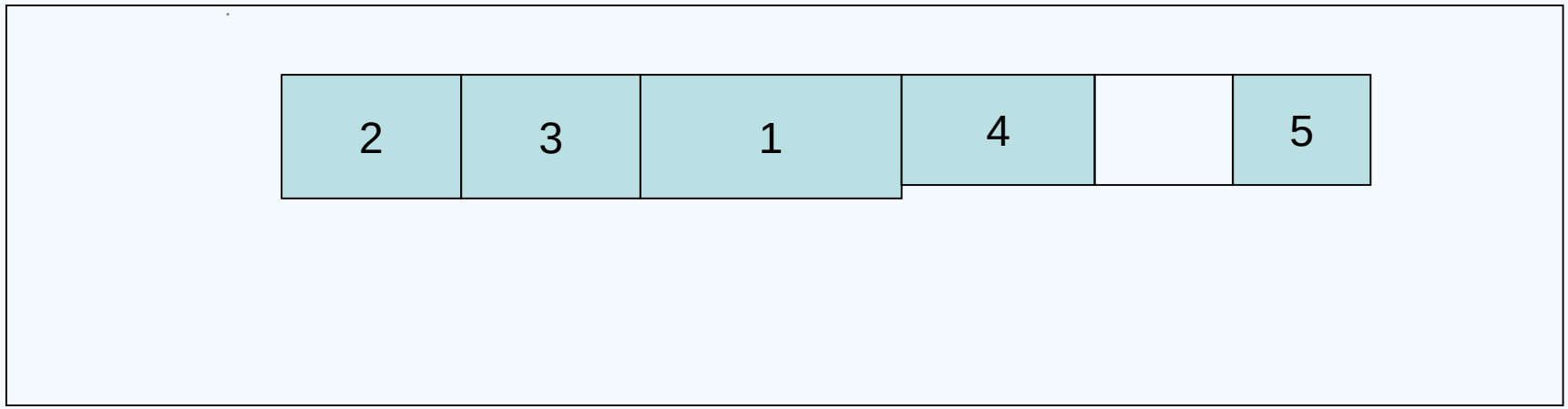
  因此，一个文档变大了，原来的位置要是放不下这个文档了，就需要把这个文档移动到集合的另外一个位置，通常是最后，能放下这个文档的地方。

- MongoDB移动文档的时候，会自动修改集合的填充因子（padding factor），填充因子是为新文档预留的增长空间，不能手动设定填充因子。

1：填充因子开始可能是1，也就是为每个文档分配精确的空间，不预留增长空间
2：当有文档超长而被迫移动文档的时候，填充因子会增大
3：当集合中不再有文档移动的时候，填充因子会慢慢减小

- MongoDB进行文档移动是非常慢的

  移动文档的时候，MongoDB需要将文档原先占用的空间释放掉，然后将文档写入新的空间，相对费时，尤其是文档比较大，又频繁需要移动的话，会严重影响性能



MongoDB的索引-1

- 创建索引，命令：ensureIndex

1：创建索引时，1表示按升序存储，-1表示按降序存储
2：可以给索引指定名字，创建的时候指定 name 即可
3：可以创建复合索引，如果想用到复合索引，必须在查询条件中包含复合索引中的前N个索引列
4：如果查询条件中的键值顺序和复合索引中的创建顺序不一致的话，MongoDB可以智能的帮助我们调整该顺序，以便使复合索引可以为查询所用
5：可以为内嵌文档创建索引，其规则和普通文档创建索引是一样的
6：一次查询中只能使用一个索引，$or特殊，可以在每个分支条件上使用一个索引
7：如果查询要在内存中排序的话，结果集不能超过32M
8：$where,$exists不能使用索引，还有一些低效率的操作符，比如：$ne,$not,$nin等
9：设计多个字段的索引时，应该尽量将用于精确匹配的字段放在索引的前面
10：MongoDB限制每个集合上最多只能有64个索引，建议在一个特定的集合上，不要设置多个索引。
11：如果要在后台运行创建索引，添加 {background:true}

- 查看已经创建的索引，命令：getIndexes
- 删除索引，命令：dropIndex
- 查看查询语句解释，命令：explain

MongoDB的索引-2

- explain的字段说明：

1：cursor：本次查询使用的索引

2：isMultiKey：是否使用了多键索引

3：n：返回的文档数量

4：nscannedObjects：按照索引指针去磁盘查找实际文档的次数

5：nscanned：如果使用索引，就是查找过的索引条目数量；如果全表扫描，就是查找过的文档数量

6：scanAndOrder：是否在内存中对结果集排序

7：indexOnly：是否只是用索引就能完成本次查询

8：nYields：为了让写入请求能顺利执行，本次查询暂停的次数

9：millis：本次查询所耗费的时间，单位是毫秒

10：indexBounds：描述索引的使用情况，给出了索引的遍历范围

- 指定使用的索引：hint
- 指定不使用索引，强制全表扫描：查询.hint({$natural:1});

MongoDB的索引-3

- 唯一索引：在创建的时候，指定{“unique”:true}

1：唯一索引将会保证该字段的数据不会重复，如果重复插入，只会保存一条

2：索引能保存的数据值必须小于1024字节，这意味着超长的数据，不会保存到索引里，因此也就可以插入多个重复的数据了

3：复合索引也可以创建为唯一索引

4：在已有集合上创建唯一索引可能会失败，因为已经有了重复数据，此时可以指定dropDups来强制去重，但由于保留数据的不可控，因此对重要数据不建议使用

5：唯一索引会把null看作值，所以无法将多个缺少唯一索引的数据插入，这时可以指定sparse来创建一个稀疏索引，如：{“unique”:true,”sparse”:true}

- 索引的集合

所有索引信息都保存在system.indexes集合众，这是一个保留集合，不能在其中进行插入或删除文档，可以通过查询集合或者使用getIndexes来查看。

Capped集合

- 简介

  Capped集合的大小固定，性能好，如果空间用完了，新的对象会覆盖旧的对象。

  find时默认就是插入的顺序，Capped集合会自动维护。
- 创建的语法：db.createCollection(集合名称,{“capped”:true,”size”:1000});

  说明：size用来指定集合大小，单位为KB
- 限制集合中对象的个数：可以在创建时设置max参数

  说明：指定max数量的时候必须同时指定size容量。淘汰机制只有在容量还没有满时才会依据文档数量来工作。要是容量满了，淘汰机制会依据容量来工作。
- 可以把已有的集合转化成为固定集合，反之不行。例如：

  db.runCommand({“convertToCapped”:”users”,”size”:1000});
- 使用和约束

1：32位机器上，一个Capped集合的最大值约为482.5M，64位只受系统文件大小的限制，创建的时候，可以预指定大小

2：可以向Capped集合中加入数据，但不能删除数据，也不能改变集合大小。

3：可以使用drop方法删除集合，删除后，需显示的重新创建这个集合

4：Capped集合在创建的时候，默认不会对任何键创建索引，如果要在”_id”上创建索引的话，在创建集合的时候，要使用autoIndexId的参数，设置为true

用GridFS存储文件-1

- 简介

  GridFS是MongoDB用来存储大型二进制文件的一种存储机制。

  特别适合用在存储一些不常改变，但是经常需要连续访问的大文件的情况。

- GridFS的优点

1：能够简化技术栈，如果已经使用了MongoDB，那么使用GridFS，就不需要其它独立的存储工具了

2：GridFS会自动平衡已有的复制，或者为MongoDB设置的自动分片，所以对文件存储做故障转移或者是横向扩展会更容易

3：GridFS的功能不错，能自动解决一些其他文件系统遇到的问题，如在同一个目录下存储大量的文件

- GridFS的缺点

1：性能较低，不如直接访问文件系统快

2：无法修改文档，如果要修改GridFS里面的文档，只能是先删除再添加

用GridFS存储文件-2

- 基本操作

最简单的就是使用自带的mongofiles工具。

1：查看文件列表：mogofiles list

2：上传文件： mogofiles put 文件路径和文件名

3：下载文件： mogofiles get 文件名

4：查找文件： mogofiles search XXX，查找所有文件名中包含XXX的文件

mogofiles list XXX ，查找所有文件名以XXX开头的文件

3：删除文件： mogofiles delete 文件名

- 基本原理

GridFS会将大文件分割为多个比较大的块，将每个快作为独立的文档进行存储，另外用一个文档来将这些块组织到一起，并存储该文件的元信息。

GridFS中的块被存储到专用的集合中，默认是fs.chunks，可以通过db.fs.chunks.find();查看这个集合，里面的结构非常简单，说明如下：

1：files_id：块所属文件的元信息

2：n：块在文件中的相对位置

3：data：块所包含的二进制数据

用GridFS存储文件-3

GridFS中每个文件的元信息被存储到单独的集合中，默认是fs.files，里面的结构非常简单，说明如下：

1：_id：文件唯一的id，就是前面的files_id

2：length：文件包含的字节数

3：chunkSize：组成文件的每个快的大小，单位是字节，这个值默认是256k

4：uploadDate：文件被上传到GridFS的日期

5：md5：文件内容的md5校验值

MongoDB的副本集-1

n 副本集简介

所谓副本集，就是指一组服务器的集群，其中有一个主服务器，用于处理用户的请求；其余为备份服务器，用于保存主服务器的数据副本。如果主服务器崩溃了，会自动将一个备份服务器升级为新的主服务器，从而保证服务的进行。

MongoDB提供复制的功能，用来将数据保存到多台服务器上，在实际生产环境中，强烈建议集群并使用复制的功能，以实现故障转移和健壮服务。

n 创建副本集

1：先创建几个存放数据的文件夹，比如在前面的dbs下面创建db1,db2,db3；
同理在前面的logs下面创建logs1,logs2,logs3

2：在启动MongoDB服务器的时候，使用--replSet 副本集名称 选项，如：

```
./mongod --dbpath ../dbs/db1 --logpath ../logs/log1 --port 20001 --fork --replSet myrepl
```

3：然后再启动两个，端口分别为20002和20003，当然要修改相应的数据文件路径和日志路径，副本集名称跟上面一样

4：连接到副本集1，进行副本集的初始化

```
rs.initiate({_id:"myrepl",members:[
  {_id:0,host:'127.0.0.1:20001'},
  {_id:1,host:'127.0.0.1:20002'},
  {_id:2,host:'127.0.0.1:20003'}]
})
```

MongoDB的副本集-2

5：察看副本集状态

rs.status(); 或者 rs.config(); 或者 db.printReplicationInfo();

6：几点说明：

（1）客户端的读写请求，都是发送到主节点进行操作

（2）客户端不能在备份节点上进行写请求

（3）默认情况下，客户端不能从备份节点读取数据，可以通过显示的执行如下语句来允许读：db.getMongo().setSlaveOk();

至此就创建好了副本集，可以进行使用测试了。

- rs辅助函数

  rs是一个全局变量，其中包含了与复制相关的辅助函数，可以通过rs.help()查看。

- rs.status()说明：

  rs.status函数对应的命令是replSetGetStatus，返回的信息中，主要的字段说明：

1：self：这个字段只会出现在运行rs.status函数的成员信息中

2：stateStr：服务器状态，状态选项在后面讲心跳的时候有

3：uptime：从成员可达一直到现在所经历的时间，单位是秒

MongoDB的副本集-3

4：optimeDate：每个成员的oplog中最后一个操作发生的时间。
注意：这里的状态是每个成员通过心跳报告上来的状态，所以optime跟实际时间可能会有几秒钟的偏差

5：lastHeartbeat：当前服务器最后一次收到其他成员心跳的时间，如果网络故障，或者是当前服务器比较繁忙，这个时间可能会是2秒钟之前

6：pingMs：心跳从当前服务器到达某个成员所花费的平均时间

7：errmsg：成员在心跳请求中返回的状态信息，通常是一些状态信息，而不是错误信息。

8：state：也表示服务器的状态，state是内部表示，而stateStr是适合阅读的表示

9：optime：和optimeDate也是一样的，只是optimeDate更适合阅读

10：syncingTo：表示当前成员正在从哪个成员处进行复制

- 修改副本集配置

1：从副本集中删除成员：rs.remove(“ip:port”)

2：为副本集添加成员：rs.add(“ip:port”);

- 副本集中主节点的确定

是通过选举机制，要大多数的节点同意的节点才能成为主节点。

MongoDB的副本集-4

- 成员配置选项——选举仲裁者

所谓仲裁者，就是不保存数据，专门用来投票选举主节点的副本，以解决副本个数为偶数的情况。

1：启动仲裁者和普通的副本方式一样

2：只是配置该节点的时候，设置：arbiterOnly:true ，如果用rs来加入的话，应该是：rs.addArb(“ip:port”);

3：最多只能有一个仲裁者

4：尽量使用奇数个成员，而不是采用仲裁者

- 成员配置选项——优先级

优先级用来表示一个成员渴望成为主节点的程度，可以在0-100之间，默认是1。

1：如果优先级为0的话，表示这个成员永远不能够成为主节点。

2：拥有最高优先级的成员会优先选举为主节点，只要它能得到“大多数”的票，并且数据是最新的，就可以了

3：如果一个高优先级的成员，数据又是最新的，通常会使得当前的主节点自动退位，让这个优先级高的做主节点

MongoDB的副本集-5

- 成员配置选项——隐藏成员

对于设置为隐藏的成员，客户端不能发送请求，也不会作为复制源，通常用来做备份服务器，方式是：

1：在配置这个节点的时候，设置hidden:true
2：只有优先级为0的成员才能被隐藏
3：可以通过rs.config()或者rs.status()查看到

- 成员配置选项——延迟备份节点

可以通过slaveDelay设置一个延迟的备份节点，以在主节点的数据不小心被破坏过后，能从备份中恢复回来。要求该备份节点的优先级为0。

- 成员配置选项——创建索引

如果不需要备份节点与主节点拥有一样的索引，可以设置：buildIndexes:false

1：这是一个永久选项，一旦指定了，就无法恢复为创建索引的正常的节点了
2：如果要恢复，只能删掉，重新再创建节点了
3：同样要求该节点的优先级为0

MongoDB副本集的基本原理-1

- 操作日志oplog

Oplog是主节点的local数据库中的一个固定集合，按顺序记录了主节点的每一次写操作，MongoDB的复制功能是使用oplog来实现的，备份节点通过查询这个集合就可以知道需要进行哪些数据的复制了。

每个备份节点也都维护着自己的oplog，记录着每次从主节点复制数据的操作。这样每个节点都可以作为数据的同步源提供给其他成员使用。

注意几点：

1：由于是先复制数据，再写日志，因此可能会出现重复的复制操作，这个没有关系，MongoDB会处理这种情况，多次执行Oplog中同一个操作与执行一次是一样的。

2：oplog的大小是固定的，它只能保存特定数量的操作日志，如果多次大批量的执行操作，oplog很快就会被填满，oplog的大小，可以通过mongod的--plogSize来指定

3：oplog的内容，在local数据库的oplog.rs集合里面

- 初始化同步

当副本集中的一个成员启动的时候，它就会检查自身状态，然后到集群中检查是否需要同步，以及从哪儿同步，并进行数据的复制，这个过程就称为初始化同步。

大致步骤如下：

MongoDB副本集的基本原理-2

1：记录前的准备工作：选择一个成员作为同步源，在local.me中为自己创建一个标识符，删除所有已存在的数据库，以全新的状态开始同步
2：从数据源把所有数据复制到本地
3：写oplog，把所有复制中的操作写入oplog
4：把上一步oplog同步中的操作记录下来
5：此时数据应该和数据源一致了，可以开始创建索引了
6：如果当前节点的数据仍然远远落后于数据源，那么oplog会将创建索引期间的所有操作同步过来
7：完成初始化同步后，切换到普通同步状态，也就可以作为备份节点使用了

- 处理陈旧数据

如果备份节点远远落后于同步源的数据，那么这个节点上的数据就是成旧数据。

通常的处理办法是：让主节点使用比较大的oplog，以保存足够多的操作日志，可以让备份节点慢慢来同步操作。

MongoDB副本集的基本原理-3

n 心跳

为了维护集合的最新状态，每个成员每隔2秒会向其他成员发送一个心跳请求，用于检查每个成员的状态。常见状态如下：

1：主节点、备份节点

2：STARTUP：成员刚启动时处于这个状态，加载完副本集配置后，进入STARTUP2

3：STARTUP2：整个初始化同步过程都处于这个状态

4：RECOVERING：表明成员运转正常，但暂时还不能处理读取请求

5：ARBITER：仲裁者始终处于这个状态

系统出问题的话，会处于如下状态：

1：DOWN：成员不可到达，有可能成员还是在正常运行，也有可能挂了

2：UNKNOWN：标识一个成员无法到达其他所有的成员

3：REMOVED：成员被移出副本集

4：ROLLBACK：成员正在进行数据回滚

5：FATAL：成员出现了致命的错误，也不再尝试恢复正常

MongoDB副本集的基本原理-4

- 回滚

回滚的时候，会把受到影响的文档，保存到数据目录下的rollback目录中，文件名为集合名.bson，基本步骤是：

1：判断是否需要回滚，典型就是副本记录和主记录不一致了

2：在两个oplog中寻找最后一个共同的点

3：然后副本回滚到这个共同的点

4：副本再从主节点同步数据回来

注意：

如果要回滚的数据很多，比如大于300M，或者需要操作30分钟以上，回滚就会失败。如果回滚失败的话，就需要重新同步了。

MongoDB副本集的管理-1

- 以单机模式启动成员

由于很多维护的工作需要写入操作，所以不合适在副本集中操作，可以以单机模式启动成员，也就是不要使用副本的选项，就跟以前启动单独的服务器一样。一般使用一个跟副本集配置中不一样的端口号，这样其他成员会认为这个服务器挂了。

- 副本集的配置

副本集的配置以一个文档的形式保存在local.system.replSet集合中，副本集中所有成员的这个文档都是相同的。绝对不要使用update来更新这个文档，应该使用rs或者replSetReconfig命令来修改副本集的配置。

1：创建副本集，这个前面学过，就是启动所有的成员服务器后，使用rs.initiate命令

2：修改副本集成员，前面也学过一些，比如rs.add()，rs.remove()命令，还可以使用rs.reconfig(config)命令，比如修改第一个成员的host名称，示例如下：

```
var config = rs.config();
config.members[0].host=“newHost:port”;
rs.reconfig(config);
```

修改其他的也一样，先rs.config()得到当前配置，然后修改数据，再rs.reconfig就OK了，但有如下几个限制：

MongoDB副本集的管理-2

（1）：不能修改成员的”_id”字段

（2）：不能将接收rs.reconfig命令的成员的优先级设置为0

（3）：不能将仲裁者变成正常成员，反之也不行

（4）：不能将buildIndexes为false的成员修改为true

- 创建比较大的副本集

  副本集最多只能拥有12个成员，只有7个成员拥有投票权。因此要创建超过7个副本集的话，需要将其他成员的投票权设置为0，例如：

  rs.add({“_id”:8,”host”:”localhost:20008”,”votes”:0});

  如果要配置超过12个成员的话，需要使用Master/Slave的方式，不过这个方式已经不建议使用了，如果将来副本集能支持更多成员的话，这个方式可能会立即废除。

- 强制重新配置

  如果副本集无法达到“大多数”要求的话，可能会无法选出主节点，这个时候，可以shell连接任意一个成员，然后使用force选项强制重新配置，示例如下：

  rs.reconfig(config,{“force”:true});

MongoDB副本集的管理-3

- 把主节点变为备份节点

  可以使用stepDown函数，可以自己指定退化的持续时间，示例如下：

  rs.stepDown(); 或者 rs.stepDown(60); //秒为单位

- 阻止选举

  如果对主节点进行维护，但不希望这段时间其他节点选举新的主节点，可以在每个备份节点上执行freeze命令，强制他们始终处于备份状态。例如：

  rs.freeze(100); //秒为单位，表示冻结多长时间

  如果在主节点上执行rs.freeze(0);，可以将退位的主节点重新变为主节点。

- 使用维护模式

  当在副本集的某个成员上执行一个非常耗时的功能的话，可以设置该成员进入维护模式，方式如下：db.adminCommand({“replSetMaintenanceMode”:true});

  要从维护模式中恢复的话，设置为false就可以了。

- 不作为复制源

  如果希望都从主节点复制数据，可以把所有的备份成员的allowChaining设置为false

MongoDB主从复制

- 简介

  主从复制是MongoDB中一种常用的复制方式。这种方式非常灵活，可用于备份、故障恢复、读扩展等。

  最基本的设置方式就是建立一个主节点和一个或者多个从节点，每个从节点要知道主节点的地址。

- 配置方式

1：主节点

启动的时候加个--master

2：从节点

启动的时候加上--slave 和 --source来指定主节点的ip和端口

- 添加和删除源

  除了可以在启动从节点的时候，制定source外，也可以在从节点的local数据库的sources集合里面添加主节点的信息，如：

  db.sources.insert({“host”:”ip:port”});

  不用了，就remove就好了

MongoDB的分片-1

- 简介

  所谓分片，指的就是把数据拆分，将其分散到不同机器上的过程。MongoDB支持自动分片，对应用而言，好像始终和一个单机的服务器交互一样。

- 分片和复制

  复制是让多台服务器拥有相同的数据副本，而分片是每个分片都拥有整个数据集的一个子集，且相互是不同的数据，多个分片的数据合起来构成整个数据集。

- Mongos

  用来执行客户端访问集群数据的路由，它维护着一个内容列表，记录了每个分片包含的数据，应用程序只要连接上它，就跟操作单台服务器一样。

- 配置服务器

  配置服务器就是普通的mongod服务器，保存整个集群和分片的元数据：集群中有哪些分片，分片的是哪些集合，以及数据块的分布。它极其重要，必须启用日志功能。

  在大型的集群中，建议配置3台配置服务器，就足够用了。启动配置服务器的方式：

1：先创建几个存放数据的文件夹，比如在前面的dbs下面创建confdb文件夹，然后在confdb下面创建confdb1,confdb2,confdb3文件夹；

同理在前面的logs下面创建conflogs文件夹

MongoDB的分片-2

2：然后分别启动这三个配置服务器，使用—configsvr指明是配置服务器，如下：

```
./mongod --configsvr --dbpath ../dbs/confdb/confdb1 --
    logpath ../logs/conflog/conflog1 --fork --port 30001
```

3：--configsvr默认的端口为27019，默认的数据目录为/data/configdb，可以使用--dbpath和--port自己定义。

4：注意不要使用--replSet选项，配置服务器不是副本集成员。Mongos会向所有的3台配置服务器发送写请求，并确保3台服务器拥有相同的数据。

- 启动mongos进程

1：

```
./mongos --configdb localhost:30001,localhost:30002,localhost:30003 --
    logpath ../logs/conflog/mongoslog --fork
```

2：可以启动任意多个mongos，通常是一个应用服务器使用一个mongos，也就是说mongos通常与应用服务器运行在一个机器上

3：mongos的默认端口是27017，可以用chunkSize来指定块的大小，默认是200M

- 将副本集转换成为分片

1：如果没有副本集，按照前面讲的创建并初始化一个；如果有一个副本集，就打开相应的服务器，把副本集运行起来

MongoDB的分片-3

2：use admin 也就是切换到使用admin的数据库

3：然后连接到mongos，把副本集转换成为分片，示例如下：

```
sh.addShard("myrepl/127.0.0.1:20001,127.0.0.1:20002");
```

不用把所有副本集的成员都写出来，mongos会自动检查整个副本集。副本集的名称myrepl就用作了分片的名称。

4：使用sh.status();察看状态，会发现整个副本集里面的服务都加入进来了。

5：注意：添加分片过后，客户端应该连接mongos进行操作，而不是连接副本集了。

6：也可以创建单mongod服务器的分片，但不建议在生产环境中使用

7：至此一个分片就创建好了，然后可以重复步骤，创建一个新的副本集，加入到分片中来

- 数据分片

需要明确指定分片的数据库和集合，MongoDB才会对数据进行自动分片。

1：对数据库启用分片

```
sh.enableSharding(“数据库名”);
```

2：然后指定分片的集合，还有分片的键，如果对已经存在的集合进行分片，那么指定的这个分片键上必须有索引；如果集合不存在，mongos会自动在分片键上创建索引。例如：

```
sh.shardCollection("rep1.users",{"userId":1});
```

MongoDB的分片-4

- 块（chunk）

  MongoDB将文档分组成为块，每个块由给定片键特定范围内的文档组成，一个块只存在于一个分片上，所以MongoDB用一个较小的表就能维护块和分片的映射关系。

  块与块间的数据不能重复，因此不能使用数组来作为片键，因为MongoDB会为数组创建多个索引条目，从而导致同一数据在多个块中出现。

  注意：块是个逻辑概念，而非物理存储实现

- 查看块信息

  块信息保存在config.chunks集合中，sh.status()里面也带有分块的信息。

  如果单个分片键可能重复的话，可以创建复合分片键，方式跟创建复合索引一样。

- 拆分块

  mongos会记录每个块中插入了多少数据，如果一个块的数据大小超出了块的大小，或者达到某个阀值(拆分点)， MongoDB会自动的拆分这个块。

  拆分的时候，配置服务器会创建新的块文档，同时修改旧的块范围。进行拆分的时候，所有的配置服务器都必须可以到达，否则不会进行拆分，此时就会造成“拆分风暴”，也就是mongos不断发起拆分请求，却无法拆分的情况。唯一的解决办法就是保证配置服务器的健康和稳定。

MongoDB的分片-5

- 均衡器（balancer）

均衡器会周期性的检查分片间是否存在不均衡，如果存在，则开始块的迁移。不均衡的表现是：一个分片明显比其他分片拥有更多的块。

1：均衡器可以是任意一个mongos

2：每隔几秒钟，mongos就会尝试成为均衡器，如果没有其他可用的均衡器，mongos就会对整个集群加锁，以防止配置服务器对集群进行修改，然后做一次均衡操作

3：均衡不会影响mongos的正常路由操作，因此对客户端没有任何影响

4：可以查看config.locks集合，看看哪一个mongos是均衡器，语句如下；

```
db.locks.findOne({"_id":"balancer"});
```

（1）_id为balancer的文档就是均衡器

（2）字段who表示当前或者曾经作为均衡器的mongos

（3）字段state表示均衡器是否在运行，0非活动，2正在均衡，1正在尝试得到锁，一般不会看到状态1

MongoDB的分片-6

- 限制分片大小

默认情况下，MongoDB会在分片间均匀的分配数据，但是，如果服务器配置严重失衡，比如某些机器配置非常高，而其他机器只是普通机器，那么就应该使用maxSize选项，来指定分片能增长到的最大存储容量，单位是MB。

例如：db.runCommand({“addShard”:”myrepl/ip:port”,“maxSize”:1000});

注意maxSize更像是一个建议而非规定，MongoDB不会从maxSize处截断一个分片，也不会阻止分片增加数据，而是会停止将数据移动到该分片，当然也可能会移走一部分数据。因此可以理解这个maxSize是一个建议或者提示。

- 注意集群对数据的影响

1：如果在从机上查询数据，就必须要可接受过时数据

2：使用集群时，不能把整个集合看成一个“即时快照”了，这个问题很严重，比如：

（1）在一个分片集合上，对数据进行计数，很有可能得到的是比实际文档多的数据，因为有可能数据正在移动复制

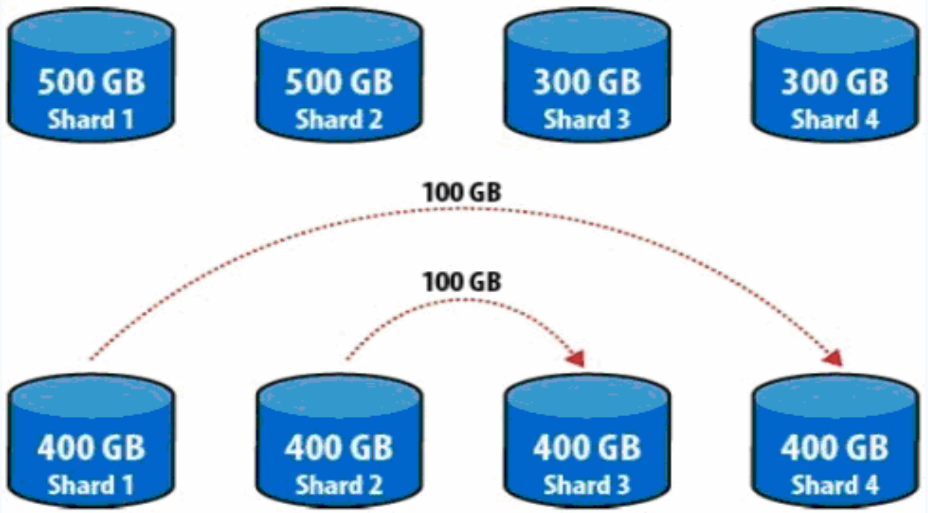
（2）唯一索引也无法强制保证了

（3）更新操作也面临问题，无法确保在多个分片间只发生一次，因此要更新单个文档，一定要在条件中使用片键，否则会出现问题

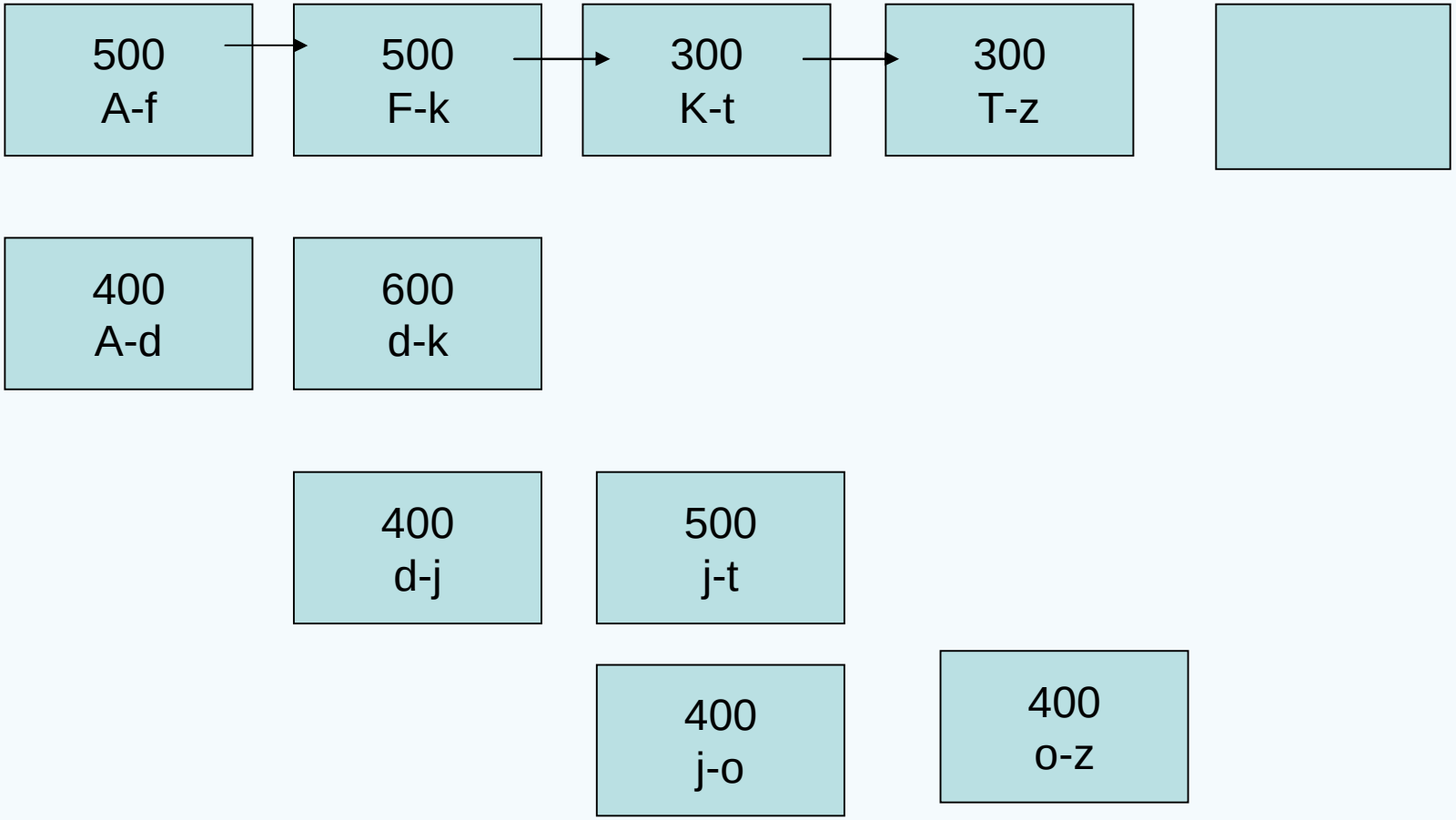
MongoDB的分片-7

- 理解MongoDB分片数据的分配方式 —— 一分片多区间

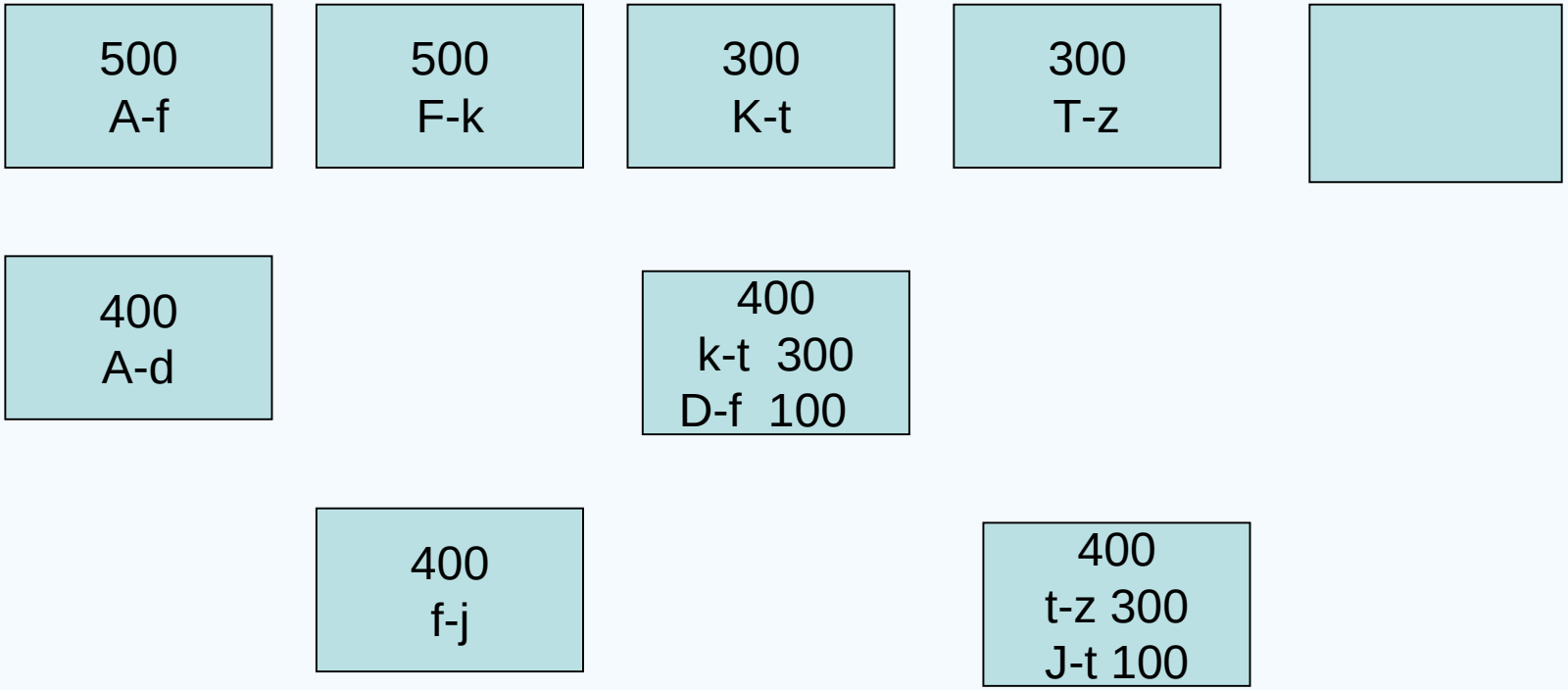
  当需要进行迁移的时候，将持有过多数据的分片上的块分割，使得分割出来的区间刚好满足迁移需要，然后再进行迁移。比如：如果shard 1中存有[a, f]区间的数据，数据量为500G，此时需要从shard 1上面迁移100G到shard 4，以保证数据的均匀分布。经统计，shard 1中的[a, d]段的数据为400G，[d, f]段数据为100G，因此将shard 1中的[d, f]段的数据直接迁移到shard 4上面。同理，需要从shard 2中迁移100G的数据到shard 3中。这种迁移方式的数据迁移量是理论上的最小值。



一分片一区间



一分片多区间



MongoDB分片的片键选择-1

- 片键选择的重要性

  所谓片键，就是用来拆分数据的字段，通常为1-2个字段，由于片键一旦确定，并已经分片过后，基本上就不可能再修改片键了，因此初期设计和选择就非常重要了

- 片键规则

  1：不可以是数组

  2：一旦插入了文档，片键不可修改，要修改就必须先删除文档，然后才能修改片键

  3：大多数特殊类型的索引都不能作为片键

  4：片键的数据取值应该是多样的，这样才利于分片

- 片键的几种类型

1：小基数片键

就是片键可取的值非常少，所以叫小基数。通常这不是个好方式，因为：片键有N个值，也就最多只能有N个块，也就是最多只能N个分片。

这也意味着当某个块越来越大的时候，MongoDB无法拆分块，因此你什么也干不了，除了购买更大的硬盘。

2：升序片键

就是片键值是不断增加的，类似于自增字段。通常这不是个好方式，因为：

MongoDB分片的片键选择-2

（1）新加入的数据始终会加入最后一个块，即所有数据都被添加到一个块上，从而导致了热点必然存在，且是单一，不可分割的热点。

（2）由于数据始终会先加入到最大块，会导致最大块需要不断的拆分出新的小块

（3）会导致数据均衡处理很困难，因为所有的新快都是由同一个分片创建的

3：随机分发的片键

就是片键值是随机散列的数据，这种方式对数据的均衡是有好处的，数据加载速度也很快，缺点是：如果需要按照片键值进行范围查找的话，就必须到所有分片上执行了。

创建一个散列片键，需要先创建散列索引，示例如下：

```
db.users.ensureIndex({“userId”:”hashed”});
```

然后对集合分片，示例如下：

```
sh.shardCollection(“mydb.users”,{“userId”:”hashed”});
```

4：基于某个业务的片键

这个就要具体问题具体分析和选择了，比如：选择用户IP，电话号码段，或者是自定义的编码段等。如果要指定特定范围的块出现在特定的分片中，可以为分片添加tag，然后为块指定tag，例如：

MongoDB分片的片键选择-3

sh.addShardTag(“myrep2”,“gtu”) 然后：
sh.addTagRange("mydb.users",{"userId":"v0"},{"userId":"v9"},"gtu");
就可以设定v0-v9的数据存放到myrep2这个分片里面去
（1）在设置范围的时候，可以使用：ObjectId()、MinKey、MaxKey 等来作为值
（2）不用某个tag了可以删除：sh.removeShardTag(“myrep2”,“gtu”);

- 好片键的建议

实际项目中，建议尽量采用 准升序键加查询键 构成组合片键。

其中升序键的每个值最好能对应几十到几百个数据块，而查询键则是应用程序通常都回依据其进行查询的字段。

例如：某应用，用户会定期访问过去一个月的数据，就可以在{month:1,user:1}上进行分片，month是一个粗粒度的升序字段，每个月都会增大；而user是经常会查询某特定用户数据的查询字段。

1：注意一点：查询键不可以是升序字段，否则该片键会退化成为一个升序片键，照样会面临热点问题

2：通常通过 准升序键 来控制数据局部化，而查询键则是应用上常用的查询字段

MongoDB分片的管理-1

- 列出所有的Shard
  可以用：db.runCommand({“listshards”:1});
- 查看分片信息
  使用函数：printShardingStatus()
- 判断是否分片
  使用：db.runCommand({isdbgrid:1});返回ok为1的就是分片
- 查看集群信息摘要
  sh.status()命令可以查看分片、数据库和分片集合的摘要信息，如果块的数量少的话，还会给出块的信息，否则它只给出集合的片键，以及每个分片的块数。
- 查看连接统计
  可使用connPoolStats命令来查看mongos和mongod之间的连接信息，如：
  db.adminCommand({"connPoolStats":1});

MongoDB分片的管理-2

- 检查配置信息

1：config.shards ： 记录着所有分片的信息
2：config.databases ：记录集群中所有数据库的信息
3：config.collections ：记录所有分片集合的信息
4：config.chunks ：记录集合中所有块的信息
5：config.changelog ：记录集群的操作
6：config.tags ：记录分片的标签

- 限制连接数量

一个mongos或者mongod最多允许20000个连接，可在mongos的命令行配置中使用maxConns选项来控制mongos能创建的连接数量。

- 添加分片服务器，这个前面学过了，就是addShard命令

MongoDB分片的管理-3

- 删除分片

通常来说，不应从集群中删除分片，即使加多了，也可以留在那儿，以后会用得上，如果非要删除分片的话，可以按照如下操作：

1：首先保证均衡器是打开的，因为删除分片的时候，均衡器会负责将待删除分片的数据迁移至其它分片

2：执行removeShard命令，示例如下：

use admin 最好切换到mongos的admin数据库再操作，然后：

```
db.runCommand({"removeShard":"myrep2"});
```

如需要查看删除情况，再次执行上一条命令，直到remaining的chunks为0。

3：所有块完成转移过后，如果仍有数据库将该分片作为主分片，需要在删除分片前将这些数据库移除掉，通常会提示：“note” ：“you need to drop or movePrimary these databases”，示例如下：

```
db.adminCommand({"movePrimary":"mydb2","to":"myrep1"});
```

4：然后再次执行removeShard命令，直到状态显示completed

监控应用状态-1

- 查看正在进行的操作

使用db.currentOp()命令，该函数会列出数据库正在进行的所有操作，其中一些字段：

1：opid：操作的唯一标识符
2：active：该操作是否正在运行
3：secs_running：该操作已经执行的时间
4：op：操作的类型
5：desc：日志中与此连接相关的每一条记录都会以[conn3]为前缀，因此可以此来筛选日志
6：locs：描述该操作使用的锁的类型，其中“^”表示全局锁
7：waitingForLock：表示该操作是否因正在等待锁而处于阻塞状态
8：numYields：表示该操作交出锁，而使其他操作得以运行的次数
9：lockstats.timeAcquiringMicros：表示该操作需要多长时间才能取得所需的锁

另外，在执行db.currentOp的时候，还可以加入过滤条件，从而只显示符合条件的结果，比如：db.currentOp({“ns”:”mydb.users”});只查询该命名空间的操作

通常用这个命令来查找耗时的操作，但是，所有跟复制、分片等相关的操作，都应该忽略掉，即使慢也不要去动，更不要去终止他们。

- 终止操作的执行

可以用db.killOp(opid);，其中的opid可以通过上面的命令来查看。

监控应用状态-2

- 系统分析器system.profile

系统分析器可记录特殊集合system.profile中的操作，并提供大量有关耗时过长的操作信息，但相应的，系统的整体性能会有所下降，因此默认是关闭的。

1：开启系统分析器：db.setProfilingLevel(级别,自定义耗时长);
设置为1级，默认会记录耗时大于100ms的操作，也可以自定义时长
设置为2级，分析器就会记录所有内容 ；
设置为0级，将会关闭系统分析器

2：通过查看system.profile集合的内容，在那个数据库上开启，就在那个数据库里面查看

- 获取服务器统计信息：db.serverStatus();
- 计算空间消耗

1：使用Object.bsonsize()来计算文档的大小，比如：
Object.bsonsize(db.users.findOne(可以加条件));

2：使用stats函数来显示一个集合的信息，比如：
db.users.stats(比例因子); 可以传入比例因子，如1024表示k，1024*1024表示M

3：使用stats函数显示数据库的信息，跟上面一个类似

- 使用mongotop和mongostat

这都是MongoDB自带的命令行工具，通过每隔几秒输出当前状态，以监控数据库

监控应用状态-3

mongotop类似于Unix的top命令。

mongostat默认每秒输出一次包含当前状态的列表，大致输出如下信息：

1：insert/query/update/delete/getmore/command ：每种操作发生的次数

2：flushes：mongod将数据flush到磁盘的次数

3：mapped：mongod所映射的内存数量，通常约等于数据目录的大小

4：vsize：mongod正在使用的虚拟内存大小，通常为数据目录的2倍，一次用于映射的文件，一次用于日志记录

5：res：mongod正在使用的内存大小，通常该值应尽量接近机器所有内存的大小

6：locked db：在上一个时间片里，锁定时间最长的数据库，该百分比是根据数据库被锁定的时间和全局锁的锁定时间来计算的，因此有可能超过100%

7：idx miss %：有所少索引在访问中发生了缺页中断，即索引入口不在内存中，使得mongod必须到磁盘进行读取

8：qr|qw：读写操作的队列大小，即有多少读写操作在阻塞队列中，等待被处理

9：ar|aw：活动客户端的数量

10：netIn：通过网络输入的字节数

11：netOut：通过网络输出的字节数

12：conn：此服务器打开的连接数

13：time：指以上统计信息的时间

用户身份验证

- 简介

MongoDB默认是没有开启身份验证的。admin和local是两个特殊的数据库，其中的用户可对任何数据库进行操作，相当于超级用户。

- 配置身份验证的方式如下：

1：首先切换到admin数据库，添加管理员用户信息，示例如下：

```
db.addUser(“root”,”cc”);
```

2：然后切换到要控制的数据库，比如mydb，然后添加用户信息，示例如下：

```
db.addUser(“u1”,”u1”); //可读写的
db.addUser(“u2”,”u2”,true); //只读的
```

3：然后重启mongod服务器，加上--auth参数，以启用安全检查

4：然后就可以测试了，验证用户使用示例：

```
db.auth(“root”,”cc”);
```

备份和恢复-1

- 文件系统快照（snapshot）

生成文件系统快照这个方法需要满足两个条件：

1：文件系统本身支持快照技术

2：运行mongod时必须开启日记系统

其恢复就是快照的恢复，当然，恢复的时候，确保mongod没有开启

- 复制文件系统

冷备份：就是把mongod停了，然后拷贝相应的数据文件，最好是把一个数据库相应的数据文件夹完整的拷出，然后恢复的时候完整的拷入。

热备份：就是mongod在运行中，可以使用db.fsyncLock();来锁定数据库，然后进行数据文件的拷贝，拷贝完成后，使用db.fsyncUnLock();解除锁定。恢复的时候，需要把mongod停了，然后把文件拷入。

- 使用mongodump来备份

这个方式有些缺点，比如速度慢，处理副本集的时候也很容易出问题。但对于单独的数据库和集合还是一个好选择的。

1：如果mongod在运行，只要指定mongod的端口即可：mongodump --port 20001，会在当前路径下创建dump文件夹，里面存放dump的数据库，真正的数据在.bson文件里。

备份和恢复-2

2：也可以-h指定主机和端口，-d指定数据库名字，-o指定输出的路经

3：如果mongod没有运行，使用--dbpath指定数据目录也可以

- 使用mongorestore来恢复

  ./mongorestore --port 20000 --drop dump/

  其中的--drop指定要删除服务器上的数据， dump/是存放备份数据的文件夹

  当然，也可以-h指定主机和端口，-d指定数据库名字，-directoryperdb指定备份文件所在的位置，也可不配置这个，直接指定就好了

- 对副本集进行备份

  建议对备份节点进行备份，最好是采用复制文件系统的方式。

- 对分片集群进行备份

  由于不可能对集群在某一时间点的完整状态快照，因此不太可能对正在运行的分片集群进行“完美的”备份。

  因此对于分片集群进行备份恢复，更关注单个分片或者副本集的备份。

  注意：要进行分片集备份，需要先关闭均衡器。

数据导入导出

- 使用mongoexport来导出数据，示例

```
./mongoexport -d mydb -c users -o ../databak/mydb.bak
```

1：-d ： 指定要导出的数据库
2：-c ： 指定要导出的集合
3：-o ： 指定输出的数据文件

还可以指定导出成csv格式的，比如：

```
./mongoexport -d mydb -c users -csv -f userId,name -o ../databak/mydb.csv
```

1：-csv ： 指定要导出成csv格式
2：-f ： 指定要导出的列

- 使用mongoimport来导入数据，示例

```
./mongoimport -d mydb -c users --file ../databak/mydb.bak
```

1：--file ： 用来指定导入的备份文件

如果要导入csv格式的话，如下：

```
./mongoimport -d mydb -c users --type csv --headerline --
file ../databak/mydb.csv
```

Java操作MongoDB-1

- 获取驱动

1：通过Maven来获取

```
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongo-java-driver</artifactId>
    <version>2.12.1</version>
</dependency>
```

2：如果不是Maven工程，也可下载mongoDB对Java支持的驱动包

地址：https://github.com/mongodb/mongo-java-driver/downloads

- 连接MongoDB服务，示例：

```
MongoClient mongo = new MongoClient("192.168.1.106",27017);
```

说明：

MongoClient内部实现了一个连接池，默认初始化10个连接，MongoClient对象是线程安全的，因此可以只创建一个，在多线程环境下安全使用。

另外要注意，close方法将关闭当前所有活跃的连接，所以应该在确定不再使用MongoDB的时候才应该关闭。

Java操作MongoDB-2

- 连接池的配置，可以修改连接池的默认配置，方法是：

1：先创建MongoClientOptions对象

2：设置该对象里面的属性值，常见的有：

（1）connectionsPerHost：每个主机的连接数

（2）threadsAllowedToBlockForConnectionMultiplier：线程队列数，它与connectionsPerHost值相乘的结果就是线程队列最大值。如果连接线程满了，就会抛出“Out of semaphores to get db”错误。

（3）maxWaitTime:最大等待连接的线程阻塞时间

（4）connectTimeout：连接超时的毫秒。默认是0，表示不超时

（5）socketTimeout：socket超时。默认是0，表示不超时

（6）autoConnectRetry：控制是否在一个连接出问题时，系统会自动重试

3：然后再连接MongoDB，此时加入选项参数即可

- 连接到MongoDB数据库，示例

```
DB db = mongo.getDB("mydb");
```

注意：

1：数据库名区分大小写，另外如果数据库不存在，不会报错，而是新建一个

2：DB对象代表了和数据库的一个连接。默认情况下，当执行完数据库的查询或者更新操作后，连接将自动回到连接池中。不需要我们手动调用代码放回池中。

- 获取要操作的集合，示例：

```
DBCollection users = db.getCollection("users");
```

Java操作MongoDB-3

- 新增

1：第一种方式，示例：

```
BasicDBObjectdata1 = new BasicDBObject();
data1.put("userId", "u1234");  data1.put("name", "name123");
//或者 data1.append("userId", "u1234").append("name", "name123");
users.insert(data1);//或者users.save(data1);
```

2：第二种方式，示例：

```
BasicDBObjectBuilder data1
=BasicDBObjectBuilder.start().add("userId","u234").add("name","name234");
users.insert(data1.get());
```

3：第三种方式，示例：

```
Map<String, Object> data1 = new HashMap<String, Object>();
data1.put("userId","u345");
users.insert(new BasicDBObject(data1));
```

4：第四种方式，示例：

```
String json = "{'userId': 'u456','name' : 'name456'}";
DBObject data1 = (DBObject)JSON.parse(json);
users.insert(data1);
```

Java操作MongoDB-4

- 删除示例

```
users.remove(new BasicDBObject("userId","u1"));
```

- 修改示例

```
DBObject condition = new BasicDBObject("userId","u234");
users.update(condition, new BasicDBObject("userId","cc").append("name","cc"));
```

当然也可以使用修改器，比如：

```
users.update(condition, new BasicDBObject("$set",new
BasicDBObject("name","ccNew")));
```

- 查询示例

1：查询全部数据

```
DBCursor dbCursor = users.find();
while(dbCursor.hasNext()){
    System.out.println(dbCursor.next());
}
```

2：查询第一个文档

```
users.findOne();
```

Java操作MongoDB-5

3：带条件查询，还可以限制返回的记录数

```
DBObject condition = new BasicDBObject("userId","u1234");
DBCursor dbCursor = users.find(condition).limit(5);
```

又比如：查询userId大于等于u1的数据，并按照userId降序排列：

```
DBObject condition = new BasicDBObject("userId",new BasicDBObject("$gte","u1"));
DBCursor dbCursor = users.find(condition).sort(new BasicDBObject("userId",-1));
```

如果要分页，在后面加上skip和limit，如：

```
users.find(condition).sort(new BasicDBObject("userId",-1)).skip(1).limit(2);
```

4：约束要返回的字段，如：

```
users.find(condition,new BasicDBObject("userId",1).append("_id",0));
```

- 其他API说明

1：创建集合，示例：db.createCollection("mycol");
2：授权认证，示例：boolean auth = db.authenticate(userName, pwd);
3：查找并删除的方法：collection.findAndRemove
4：获取所有数据库的名字的方法：mongo.getDatabaseNames()
5：获取所有集合的名字的方法：db.getCollectionNames();
6：在一个连接上执行多个操作，可以用：

```
db.requestStart();  结束的时候用  db.requestDone();
```

。。。还有很多，看API吧

MongoDB和Spring-1

- Spring通过Spring Data MongoDB模块来集成和支持MongoDB
- 加入lib包，在Maven中：

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-commons</artifactId>
    <version>1.5.1.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-mongodb</artifactId>
    <version>1.4.2.RELEASE</version>
</dependency>
```

- 在Spring的配置文件中，加入如下的命名空间

xmlns:mongo=http://www.springframework.org/schema/data/mongo

对应的location是：

http://www.springframework.org/schema/data/mongo

http://www.springframework.org/schema/data/mongo/spring-mongo-1.5.xsd

MongoDB和Spring-2

然后加入如下的Bean配置：

```
<mongo:mongo host="192.168.1.106" port="20000"></mongo:mongo>
<bean id="mongoTemplate"
    class="org.springframework.data.mongodb.core.MongoTemplate">
      <constructor-arg ref="mongo"/>
      <constructor-arg name="databaseName" value="db2"/>
</bean>
```

MongoDB应用建议及最佳实践-1

- 尽量在64位机器上使用MongoDB，当然32位机器也能用，但由于MongoDB使用内存映射文件，32位版本只支持2G数据的存储
- 避免存储过大的文件，MongoDB限制单个文档是16M
- 弱化数据结构模型并不等于没有数据结构模型，同样要合理设计
- MongoDB没有Join语句，因此要合理的设计数据结构
- 复制集成员尽量使用奇数个，如果是偶数个，可以用仲裁者凑数
- 最好启动Journal

  MongoDB使用内存映射文件并且每60秒向磁盘输出一次通知，这就意味着最大程度上你可能丢失60秒加上向硬盘输出通知这段时间内所有的数据。

  为了避免数据丢失，MongoDB从2.0版本起就添加了Journaling（默认情况下开启）。Journaling把时间从60秒更改为100ms。如果数据库意外的停机，在启动之前它将会被重启用以确保数据库处于一致状态。当然Journaling会轻微的影响到性能，大约5%。
- 单机使用MongoDB，应启用日志，就是journal；而多机使用就做副本集

MongoDB应用建议及最佳实践-2

n 关于范式化设计和反范式化设计

1：范式化设计通常是通过设计多个集合，然后用字段的值来关联数据的方式；而反范式化设计通常是把相关数据做成内嵌文档，嵌入到文档数据中

2：通常范式化设计能提高数据的写入速度，而反范式化设计能提高查询速度

3：通常认为适合使用内嵌数据的地方：

（1）子文档较小

（2）数据不会经常变化

（3）最终数据一致即可，对中间数据是否一致不要求

（4）需要快速读取的地方

4：通常认为适合使用引用数据的地方：

（1）子文档较大

（2）数据经常改变

（3）数据一致性要求高，包括中间数据都要求一致

（4）需要快速写入的地方

（5）适应未来的数据

5：可以混合使用引用数据和内嵌数据的方式

MongoDB应用建议及最佳实践-3

- 关于数据集合之间的关系

  集合之间的常见关系包括：一对一、一对多和多对多。

  在MongoDB中，又把多 按照数据量的大小，分成了“多”和“少”，通常少的关系使用内嵌的方式较好，多的关系使用引用的方式较好。

  对于多对多的处理，通常还是会引入一个关系集合，从而拆分成两个一对多。

- 优化文档增长

  如果文档中有需要不断增长的字段，那么建议把这个字段放在最后。

  如果能够预估这个字段最大的长度，还可以采用手动分配的方式，在创建文档的时候就预留足够的空间，从而避免后面移动文档，方式如下：

1：创建文档的时候，在这个字段后面，任意添加一个字段，比如：needDelete，然后给needDelete字段赋值，值任意，主要是长度为要增长的字段的最大长度

2：在后面更新文档的时候，就可以使用$unset来移出needDelete字段，如果字段存在就移出，不存在什么都不做

- 用数组存放要匿名访问的内嵌数据

  如果确切的知道存放数据的含义，并要根据含义进行查找，用文档，否则用数组

MongoDB应用建议及最佳实践-4

- MongoDB的索引代价很大
  1：索引非常消耗内存，大约100万条索引要占到50M以上
  2：索引对写入性能有较大的影响，索引多了，写入会非常慢
  因此只有当第二个索引的查询不可避免，才值得增加额外索引。
- 查询的时候，应该尽量只返回需要的数据和字段，避免无谓的流量损耗
- 如果需要查询的数据超过数据的一半，就不适合使用索引了，可以用{$natural:1}来禁用
- 更新数据的时候，应该尽量精细化更新，尽量准确的条件，并使用修改器直接修改要改变的字段，不要整条替换
- 不要使用大量的、或是复杂的js计算，对cpu性能损耗较大
- 尽早分片，一般要在MongoDB占用服务器整体性能80%前分片
- 谨慎选择分片键，一旦正式使用，几乎不能再更改了
- 不可以对256G以上的集合进行分片

MongoDB应用建议及最佳实践-5

- 文档对象数据应该能直接满足应用的需要

  MongoDB对数据基本是不做任何处理的，仅仅存取数据，因此，要获取的信息如果无法从文档中直接获取的话，就只有：

1：在MongoDB中使用js计算，会付出高昂的性能代价

2：把大量数据取到客户端，然后在客户端进行计算

两种方式都不会太好，因此要合理的设计文档对象，最好能直接从文档中获取应用需要的数据

- 不要用GridFS处理小的二进制数据

  由于GridFS需要二次查询，一次获取文件元数据，一次获取其内容，因此，GridFS更适合存放大数据的，至少是一个文档存放不下的数据，通常客户端也不用一次性加载这些数据。

- And型查询，应该把条件最严格的条件放在前面
- OR型查询，应该把最有可能为true的条件放在前面