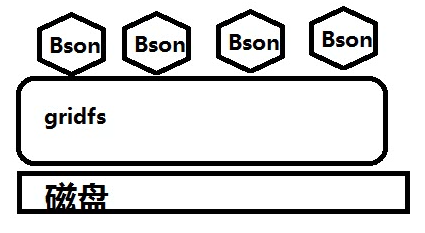
**mogodb学习笔记**

官网: https://www.mongodb.org

mogodb是文档数据库，存储的是文档(Bson，json的二进制数据)，内部执行引擎为JS解析器(谷歌v8)，把文档储存成bson结构，在查询是转换为JS对象，可以通过熟悉的JS语法来操作。储存结构图如下：



mogodb和传统数据库区别：

传统数据库：结构化数据，设计了表结构后，每一行内容的列的类型是固定的。

mogodb文档数据库：表下的每篇文档都可以有自己独立的数据结构(json对象都可以有自己独特的属性和值)

# 1 在centos下安装mongodb

|  |  |
| --- | --- |
| 下载地址 | https://www.mongodb.org/dl/linux/x86\_64 |
| 下载 | wget http://downloads.mongodb.org/linux/mongodb-linux-x86\_64-rhel70-3.5.12.tgz?\_ga=2.135958961.1891202371.1512871774-1596522567.1512400407 |
| 解压 | tar zxvf mongodb-linux-x86\_64-rhel70-3.5.12.tgz  不需要编译，本身是二进制可执行文件 |
| 移到/usr/local下 | mv mongodb-linux-x86\_64-rhel70-3.5.12 mongodb  mv mongodb /usr/local |
| 添加mongodb环境变量 | export PATH=/usr/local/mongodb/bin:$PATH && source /etc/profile |
| 安装最新版的openssl | yum install openssl |

bin目录下可执行文件解释：

|  |
| --- |
| ├── **bsondump** // 导出bson结构  ├── **install\_compass**  ├── **mongo** // 客户端  ├── **mongod** // 服务端  ├── **mongodump** // 整体数据库导出(二进制)  ├── **mongoexport** // 导出易识别的json文档或csv文档  ├── **mongofiles**  ├── **mongoimport**  ├── **mongoperf**  ├── **mongoreplay**  ├── **mongorestore** // 数据库整体导入  ├── **mongos** // 路由器(分片时用)  ├── **mongostat** // 观察状态  └── **mongotop** // 类似linux的top命令 |

mongodb启动

**mongod --dbpath /data/mongodb/database --logpath /data/mongodb/log/mongodb.log --port 27017 --fork**

--dbpath：数据库储存目录，--logpath日志储存目录，--fork后台运行，--port运行端口

使用mongod --help查看有哪些参数

mongodb客户端连接：

**mongodb**

查看当前mongodb使用版本

**db.version();**

# 2 创建管理员和用户

创建管理员

|  |
| --- |
| (1)创建用户管理员(在管理身份验证数据库)。  use admin  db.createUser(  {  user: "vison",  pwd: "123456",  roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]  }  )  (2)重新启动MongoDB实例与访问控制。  mongod --dbpath /data/mongodb/database --logpath /data/mongodb/log/mongodb.log --port 27017 --fork --auth  (3)连接和用户管理员进行身份验证。  mongo --port 27017 -u "vison" -p "123456" --authenticationDatabase "admin" |

创建普通用户

|  |
| --- |
| (1)根据需要创建额外的用户为您的部署(在测试验证数据库)。  use test  db.createUser(  {  user: "myTester",  pwd: "xyz123",  roles: [ { role: "readWrite", db: "test" },  { role: "read", db: "reporting" } ]  }  )  (2)mytest连接和验证。  mongo --port 27017 -u "myTester" -p "xyz123" --authenticationDatabase "test" |

# 3 mongodb入门命令

## 库级命令

查看所有数据库

**show dbs** // 或使用mysql命令show databases;

切换到指定数据库

**use 库名**

创建数据库(mongodb不提供命令)

mongodb的数据库是隐式创建的(创建表时自动创建)，就算没有指定的库，也可以使用use命令。

删除数据库

**db.dropDatabase()**

## 表(集合)级命令

显示所有表

**show collections;** //或mysql命令show tables;

创建表

**db.createCollection('user')**

删除表

**db.user.drop()**

## 3.1 增 insert

### 3.1.1 增加一个文档

不指定id

**db.user.insert({name:'张三',age:22})**

指定id

**db.user.insert({\_id:10001,name:'李四',age:23})**

### 3.1.2 一次增加多个文档

**db.user.insert([{name:'王五',age:24},{\_id:10002,name:'赵六',age:25},{name:'刘七',age:26}])**

## 3.2 删 remove

语法：db.collection.remove(查询表达式,选项)

查询表达式:类似mysql的where条件，不加条件会删除表所有数据。

选项: 是否删除一行，{justOne:true/false}，默认是false

**db.user.remove({\_id:10001})**

**db.user.remove({age:24},{ justOne:true})**

实际应用中一般不使用删除，使用一个字段来标记删除。

## 3.3 改 update

语法：db.collection.updae(查询表达式,新值,选项)

**~~db.user.update({name:'张三'},{age:33})~~**

注意：这种方式是新文档直接替换了旧文档，必须指定赋值表达式，如图所示



### 3.3.1 赋值表达式

$set设置字段新值(如果字段不存在则新增加该列):

**db.user.update({name:'张三'},{$set:{age:33}})**

$inc自增字段值：

**db.user.update({name:'张三'},{$inc:{age:1}})**

$rename修改字段名：

**db.user.update({name:'张三'},{$rename:{sex:'gender'}})**

$unset删除指定列：

**db.user.update({name:'张三'},{$unset:{phone:1}})**

$setOnInsert当upsert为true时，添加附加字段：

**db.user.update({name:'赵六'},{$setOnInsert:{age:25,gender:'male'}},{upsert:true})**

多种赋值表达可以一次执行db.collection.updae(查询表达式,{

{$set:{...}},

{$unset:{...}},

{$inc:{...}},

{$rename:{...}}

})

### 3.3.2 修改选项

选项包括：{upsert:true/false,multi:true/false}

upsert：没有匹配的行，是否直接插入该行，默认为false

**db.user.update({name:'李四'},{$set:{age:44}},{upsert:true})**

multi：查询表达式匹配中多行，是否修改多行，默认false（只修改一行）

**db.user.update({gender:'male'},{$set:{age:55}},{multi:true})**

## 3.4 查 query

语法：db.collection.find(查询表达式,查询的列)

**db.user.find()** // 查询所有

查询匹配条件的所有列：

**db.user.find({name:'张三'})**

查询匹配条件的指定列({列名:1,列名:1,...})：

**db.user.find({name:'张三'},{age:1})**

不查询id列

**db.user.find({name:'张三'},{\_id:0,age:1})**

有goods和category两个表

### 3.4.1 字段值查询

查询主键为32的商品

**db.goods.find({goods\_id:32})**

子文档查询(指向子文档的key是字符串)

**db.stu.find({'score.yuwen':75})**

数组元素查询(指向子文档的key是字符串)

**db.stu.find({'hobby.2':'football'})**

### 3.4.2 范围查询

(1) $ne 不等于

查询不属第3栏目的所有商品

**db.goods.find({cat\_id:{$ne:3}})**

(2) $gt 大于

查询高于3000元的商品

**db.goods.find({shop\_price:{$gt:3000}})**

(3) $gte 大于等于

查询高于或等于3000元的商品

**db.goods.find({shop\_price:{$gte:** **3700}})**

(4) $lt 小于

查询低于500元的商品

**db.goods.find({shop\_price:{$lt:500}})**

(5) $lt 小于等于

查询低于500元的商品

**db.goods.find({shop\_price:{$lte:280}})**

### 3.4.3 集合查询

(1) $in 在集合，(当查询字段是数组时，不需完全匹配也会命中该行)

查询栏目为4和11的商品

**db.goods.find({cat\_id:{$in:[4,11]}})**

(2) $nin 不在集合

查询栏目不是3和4的商品

**db.goods.find({cat\_id:{$nin:[3,4]}})**

(3) $all完全匹配，(当查询字段是数组时，必须完全匹配才命中该行)

**db.goods.find({cat\_id:{$all:[4,11]}})**

### 3.4.4 逻辑查询

使用表达式拼接各条件

(1) $and 逻辑与，必须都满足条件

查询价格为100到500的商品

**db.goods.find({$and:[{shop\_price:{$gt:100}},{shop\_price:{$lt:500}}]})**

(2) $or 逻辑或，至少满足一个条件

查询价格小于100或大于3000的商品

**db.goods.find({$or:[{shop\_price:{$lt:100}},{shop\_price:{$gt:3000}}]})**

(3) $nor 与非逻辑

查询不在栏目3，并且价格不小1000的商品

**db.goods.find({$nor:[{cat\_id:3},{shop\_price:{$lt:1000}}]})**

### 3.4.5 元素运算符查询

(1) $mod 求余

查询年龄对10求余为0的用户

**db.goods.find({age:{$mod:[10,0]}})**

(2) $exist 查询列是否存在

查询有电话属性的用户

**db.user.find({phone:{$exists:1}})**

(3) $type 查询属性为指定类型的文档

查询为浮点型的文档

**db.user.find({phone:{$type:1}})**

查询为字符串型的文档

**db.user.find({phone:{$type:2}})**

### 3.4.6 js运算查询

把存在磁盘的二进制文档转为js格式，在通过js解析器判断，所以效率比较低下，不是非必要，一般不使用。

(1) $where 条件查询，类似mysql的where

查询价格在100到500，或3000到5000的商品

**db.goods.find({$where:'(this.shop\_price>100&& this.shop\_price<500)||(this.shop\_price>3000&& this.shop\_price<5000)'})**

(2) $regex 正则匹配查询

查询名称为诺基亚开头的商品

**db.goods.find({goods\_name:{$regex:/^诺基亚.\*/}})**

### 3.4.7 其他常用查询

(1) 统计行数

**db.goods.count()**

**db.goods.find({cat\_id:3}).count()** // 栏目id为3的商品数

(2) 查询每页数据

sort({属性:1|-1}) // 按指定属性排序，1表示升序，-1表示降序

skip(num) // 跳过行数

limit(num) // 限制取多少行

页码page，每页行数n，则skip数=(page-1)\*n

**db.goods. find({cat\_id:3}).sort({shop\_price:-1}).skip(0).limit(10)**

# 4 索引

索引提高查询速度，降低写入速度，权衡常用的查询字段，不建议在太多列上建索引。

在mongodb中，索引可以按字段升序/降序来创建，便于排序

默认是用btree来组织索引文件，也允许建立hash索引。

## 4.1 查看索引和执行计划

查看表索引

**db.stu.getIndexes()**

查看执行计划

**db.stu.find({sn:5555}).explain()** // 默认只输出queryPlanner

其中explain()参数有三个，分别是'queryPlanner'、'executionStats'、'allPlansExecution'

**db.stu.find({sn:5555}).explain('executionStats')**

explain分析结果的几个重要字段，通过结果分析可以判断是否需要优化执行语句

|  |  |
| --- | --- |
| executionStats属性下的字段 | |
| executionTimeMillis | 查询耗时，单位(ms) |
| totalDocsExamined | 扫描文档数 |
| executionStages.stage | "COLLSCAN"表示全表扫描  " FETCH"表示索引扫描 |
| executionStages.  executionTimeMillisEstimate | 索引扫描耗时，单位(ms) |
| winningPlan.inputStage属性下的字段 | |
| indexName | 索引名字 |

## 4.2 普通索引

首先在test库下stu下创建10000行数据

|  |
| --- |
| for (var i=1;i<=10000;i++){db.stu.insert({sn:i,name:'stu'+i,email:'stu'+i+'@126.com',score:{yuwen:i%80,shuxue:i%90,yingyu:i%100}})} |

### 4.2.1 单列索引

在表stu创建sn列索引

**db.stu.ensureIndex({sn:1})** // 1表示升序，-1表示降序

### 4.2.2 多列索引

在表stu创建sn列和name列共同索引

**db.stu.ensureIndex({sn:1,name:1})** // 1表示升序，-1表示降序

### 4.2.3 子文档索引

在表stu的score列下的yuwen字段创建索引

**db.stu.ensureIndex({'score.yuwen':1})** // 1表示升序，-1表示降序

## 4.3 唯一索引

创建唯一索引后字段值都是唯一的

在表stu创建email列索引

**db.stu.ensureIndex({email:1},{unique:true})**

## 4.4 稀疏索引

稀疏索引的特点：如果针对field做索引，针对不含field列的文档，将不建立索引。

与之相对的普通索引会把该文档的field列的值认为NULL，并建索引。

使用场景：小部分文档含有某列时。

在表stu创建phone列稀疏索引

**db.stu.ensureIndex({age:1},{sparse:true})**

## 4.5 哈希索引

哈希索引速度比普通索引快，缺点是不能对范围查询进行优化。

使用场景：随机性强的散列

在表stu创建email列哈希索引

**db.stu.ensureIndex({email:'hashed'})**

## 4.6 重建索引

一个表经过很多次修改后，导致表的文件产生空洞，索引文件也如此。

可以通过索引的重建，减少索引文件碎片，并提高索引的效率，类似mysql中的optimize table

在表stu重建索引

**db.stu.reIndex()**

## 4.7 删除索引

语法：db.collection.dropIndex({filed:1/-1});

把创建时的**ensureIndex改为dropIndex即可**

**db.stu.dropIndex({sn:1})**

**db.stu.dropIndex ({email:'hashed'})**