MONGODB

mongo课程概要

1. mongodb的介绍
2. mongodb安装
3. mongodb基本操作
4. mongodb数据查询
5. mongdb聚合
6. 索引和备份
7. mongo和python交互

nosql的介绍

- “NoSQL”一词最早于1998年被用于一个轻量级的关系数据库的名字
- 随着web2.0的快速发展， NoSQL概念在2009年被提了出来
- NoSQL在2010年风生水起， 现在国内外众多大小网站， 如facebook、google、 淘宝、 京东、 百度等， 都在使用nosql开发高性能的产品
- 对于一名程序员来讲， 使用nosql已经成为一条必备技能
- NoSQL最常见的解释是“non-relational”， “Not Only SQL”也被很多人接受， 指的是非关系型的数据库

关系型和非关系型的介绍

```
{    customer_id : 1,
     first_name : "Mark",
     last_name : "Smith",
     city : "San Francisco",
     phones: [ {
          type : "work",
          number: "1-800-555-1212"
     },
     {    type : "home",
          number: "1-800-555-1313",
          DNC: true
     },
     {    type : "home",
          number: "1-800-555-1414",
          DNC: true
     }
   ]
}
```

关系型和非关系型的介绍

Relational

关系数据库很强大，但是它并不能很好的应付所有的应用场景。
MySQL的扩展性差，大数据下IO压力大， 表结构更改困难

MongoDB

易扩展，大数据量高性能，灵活的数据模型，高可用

mongdb的优势

- 易扩展： NoSQL数据库种类繁多， 但是一个共同的特点都是去掉关系数据库的关系型特性。 数据之间无关系， 这样就非常容易扩展
- 大数据量， 高性能： NoSQL数据库都具有非常高的读写性能， 尤其在大数据量下， 同样表现优秀。 这得益于它的无关系性， 数据库的结构简单
- 灵活的数据模型： NoSQL无需事先为要存储的数据建立字段， 随时可以存储自定义的数据格式。 而在关系数据库里， 增删字段是一件非常麻烦的事情。 如果是非常大数据量的表， 增加字段简直就是一个噩梦

mongodb安装

sudo apt-get install -y mongodb-org

https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/

mongodb安装

- 解压

```
tar -zxvf mongodb-linux-x86_64-ubuntu1604-3.4.0.tgz
```

- 移动到/usr/local/目录下

```
sudo mv -r mongodb-linux-x86_64-ubuntu1604-3.4.0/ /usr/local/mongodb
```

- 将可执行文件添加到PATH路径中

```
export PATH=/usr/local/mongodb/bin:$PATH
```

服务端mongodb启动

查看帮助：mongod –help

启动：sudo service mongod start

停止：sudo service mongod stop

重启：sudo service mongod restart

查看是否启动成功：ps ajx|grep mongod

配置文件的位置：/etc/mongod.conf，

默认端口：27017

日志的位置：

/var/log/mongodb/mongod.log

客户端mongo

启动本地客户端:mongo

查看帮助：mongo –help

退出：exit或者ctrl+c

关于database的基础命令

查看当前的数据库：db
查看所有的数据库：show dbs /show databases
切换数据库：use db_name
删除当前的数据库：db.dropDatabase()

关于集合的基础命令

不手动创建集合：

向不存在的集合中第一次加入数据时， 集合会被创建出来

手动创建结合：

db.createCollection(name,options)

db.createCollection("stu")

db.createCollection("sub", { capped : true, size : 10 })

参数capped： 默认值为false表示不设置上限,值为true表示设置上限

参数size： 当capped值为true时， 需要指定此参数， 表示上限大小,当文档达到上限时， 会将之前的数据覆盖， 单位为字节

查看集合：show collections

删除集合：db.集合名称.drop()

数据类型

Object ID： 文档ID
String： 字符串， 最常用， 必须是有效的UTF-8
Boolean： 存储一个布尔值， true或false
Integer： 整数可以是32位或64位， 这取决于服务器
Double： 存储浮点值
Arrays： 数组或列表， 多个值存储到一个键
Object： 用于嵌入式的文档， 即一个值为一个文档
Null： 存储Null值
Timestamp： 时间戳， 表示从1970-1-1到现在的总秒数
Date： 存储当前日期或时间的UNIX时间格式

注意点

- 创建日期语句如下： 参数的格式为YYYY-MM-DD
  new Date('2017-12-20')

- 每个文档都有一个属性， 为_id， 保证每个文档的唯一性
- 可以自己去设置_id插入文档，如果没有提供， 那么MongoDB为每个文档提供了一个独特的_id， 类型为objectID

- objectID是一个12字节的十六进制数:
  前4个字节为当前时间戳
  接下来3个字节的机器ID
  接下来的2个字节中MongoDB的服务进程id
  最后3个字节是简单的增量值

插入

- db.集合名称.insert(document)
- db.stu.insert({name:'gj',gender:1})
- db.stu.insert({_id:"20170101",name:'gj',gender:1})
- 插入文档时， 如果不指定_id参数， MongoDB会为文档分配一个唯一的ObjectId

保存

- db.集合名称.save(document)
- 如果文档的_id已经存在则修改， 如果文档的_id不存在则添加

简单查询

db.集合名称.find()

更新

db.集合名称.update(<query> ,<update>,{multi: <boolean>})

参数query:查询条件
参数update:更新操作符
参数multi:可选， 默认是false，表示只更新找到的第一条记录，值为true表示把满足条件的文档全部更新

db.stu.update({name:'hr'},{name:'mnc'}) 更新一条
db.stu.update({name:'hr'},{$set:{name:'hys'}}) 更新一条
db.stu.update({},{$set:{gender:0}},{multi:true}) 更新全部

注意："multi update only works with $ operators"

删除

db.集合名称.remove(<query>,{justOne: <boolean>})

参数query:可选，删除的文档的条件
参数justOne:可选， 如果设为true或1， 则只删除一条， 默认false， 表示删除多条

数据查询

- 方法find()： 查询

  db.集合名称.find({条件文档})

- 方法findOne()：查询，只返回第一个

  db.集合名称.findOne({条件文档})

- 方法pretty()： 将结果格式化

  db.集合名称.find({条件文档}).pretty()

比较运算符

- 等于： 默认是等于判断， 没有运算符
- 小于：$lt （less than）
- 小于等于：$lte （less than equal）
- 大于：$gt （greater than）
- 大于等于：$gte
- 不等于：$ne

db.stu.find({age:{$gte:18}})

逻辑运算符

- and：在json中写多个条件即可

查询年龄大于或等于18， 并且性别为true的学生

```
db.stu.find({age:{$gte:18},gender:true})
```

- or:使用$or， 值为数组， 数组中每个元素为json

查询年龄大于18， 或性别为false的学生

```
db.stu.find({$or:[{age:{$gt:18}},{gender:false}]})
```

- 查询年龄大于18或性别为男生， 并且姓名是郭靖

```
db.stu.find({$or:[{age:{$gte:18}},{gender:true}],name:'gj'})
```

范围运算符

使用"$in"， "$nin" 判断是否在某个范围内
查询年龄为18、 28的学生
db.stu.find({age:{$in:[18,28]}})

支持正则表达式

使用//或$regex编写正则表达式
查询姓黄的学生
db.stu.find({name:/^黄/})
db.stu.find({name:{$regex:'^黄'}})

limit和skip

- 方法limit()： 用于读取指定数量的文档

db.集合名称.find().limit(NUMBER)
查询2条学生信息
db.stu.find().limit(2)

- 方法skip()： 用于跳过指定数量的文档

db.集合名称.find().skip(NUMBER)
db.stu.find().skip(2)

- 同时使用

db.stu.find().limit(4).skip(5)
或
db.stu.find().skip(5).limit(4)

自定义查询*

使用$where后面写一个函数， 返回满足条件的数据

查询年龄大于30的学生

```
db.stu.find({
    $where:function() {
        return this.age>30;}
})
```

投影

在查询到的返回结果中， 只选择必要的字段
db.集合名称.find({},{字段名称:1,...})
参数为字段与值， 值为1表示显示， 值为0不显
特殊： 对于_id列默认是显示的， 如果不显示需要明确设置为0

db.stu.find({},{_id:0,name:1,gender:1})

排序

方法sort()， 用于对集进行排序
db.集合名称.find().sort({字段:1,...})
参数1为升序排列
参数-1为降序排列
根据性别降序， 再根据年龄升序
db.stu.find().sort({gender:-1,age:1})

统计个数

方法count()用于统计结果集中文档条数
db.集合名称.find({条件}).count()
db.集合名称.count({条件})
db.stu.find({gender:true}).count()
db.stu.count({age:{$gt:20},gender:true})

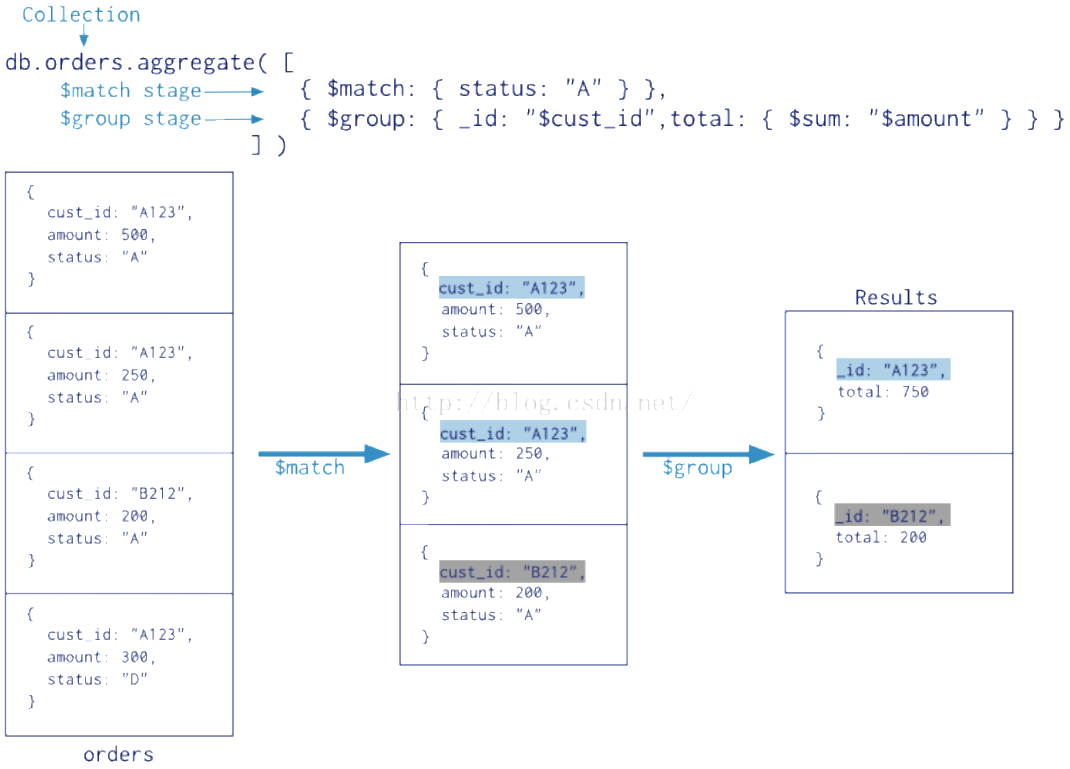
消除重复

方法distinct()对数据进行去重
db.集合名称.distinct('去重字段',{条件})
db.stu.distinct('hometown',{age:{$gt:18}})

聚合 aggregate

聚合(aggregate)是基于数据处理的聚合管道，每个文档通过一个由多个阶段（stage）组成的管道，可以对每个阶段的管道进行分组、过滤等功能，然后经过一系列的处理，输出相应的结果。

db.集合名称.aggregate({管道:{表达式}})



常用管道

在mongodb中，文档处理完毕后， 通过管道进行下一次处理
常用管道如下：
$group： 将集合中的文档分组， 可用于统计结果
$match： 过滤数据， 只输出符合条件的文档
$project： 修改输入文档的结构， 如重命名、 增加、 删除字段、 创建计算结果
$sort： 将输入文档排序后输出
$limit： 限制聚合管道返回的文档数
$skip： 跳过指定数量的文档， 并返回余下的文档
$unwind： 将数组类型的字段进行拆分

表达式

处理输入文档并输出
语法：表达式:'$列名'
常用表达式:
$sum： 计算总和， $sum:1 表示以一倍计数
$avg： 计算平均值
$min： 获取最小值
$max： 获取最大值
$push： 在结果文档中插入值到一个数组中
$first： 根据资源文档的排序获取第一个文档数据
$last： 根据资源文档的排序获取最后一个文档数据

$group

- 将集合中的文档分组，可用于统计结果
- _id表示分组的依据，使用某个字段的格式为'$字段'
- 例1：统计男生、女生的总人数

```
db.stu.aggregate(
    {$group:
        {
            _id:'$gender',
            counter:{$sum:1}
        }
    }
)
```

group文档：
https://docs.mongodb.com/manual/reference/operator/aggregation/group/

Group by null

- 将集合中所有文档分为一组
- 例2：求学生总人数、平均年龄

```
db.stu.aggregate(
    {$group:
        {
            _id:null,
            counter:{$sum:1},
            avgAge:{$avg:'$age'}
        }
    }
)
```

透视数据

统计不同性别的学生姓名

```
db.stu.aggregate(
   {$group:
      {
          _id:'$gender',
          name:{$push:'$name'}
      }
   }
)
```

使用$$ROOT可以将文档内容加入到结果集的数组中

```
db.stu.aggregate(
     {$group:
          {
               _id:'$gender',
               name:{$push:'$$ROOT'}
          }
     }
)
```

动手

{ "country" : "china", "province" : "sh", "userid" : "a" }
{ "country" : "china", "province" : "sh", "userid" : "b" }
{ "country" : "china", "province" : "sh", "userid" : "a" }
{ "country" : "china", "province" : "sh", "userid" : "c" }
{ "country" : "china", "province" : "bj", "userid" : "da" }
{ "country" : "china", "province" : "bj", "userid" : "fa" }

需求：统计出每个country/province下的userid的数量（同一个userid只统计一次）

$match

- 用于过滤数据，只输出符合条件的文档
- 使用MongoDB的标准查询操作
- 例1：查询年龄大于20的学生

match是管道命令，能将结果交给后一个管道，但是find不可以

```
db.stu.aggregate(
    {$match:{age:{$gt:20}}}
)
```

- 例2：查询年龄大于20的男生、女生人数

```
db.stu.aggregate(
    {$match:{age:{$gt:20}}},
    {$group:{_id:'$gender',counter:{$sum:1}}}
)
```

$project

- 修改输入文档的结构，如重命名、增加、删除字段、创建计算结果
- 例1：查询学生的姓名、年龄

```
db.stu.aggregate(
    {$project:{_id:0,name:1,age:1}}
)
```

- 例2：查询男生、女生人数，输出人数

```
db.stu.aggregate(
    {$group:{_id:'$gender',counter:{$sum:1}}},
    {$project:{_id:0,counter:1}}
)
```

动手练习

{ "country" : "china", "province" : "sh", "userid" : "a" }
{ "country" : "china", "province" : "sh", "userid" : "b" }
{ "country" : "china", "province" : "sh", "userid" : "a" }
{ "country" : "china", "province" : "sh", "userid" : "c" }
{ "country" : "china", "province" : "bj", "userid" : "da" }
{ "country" : "china", "province" : "bj", "userid" : "fa" }

需求：统计出每个country/province下的userid的数量（同一个userid只统计一次），结果中的字段为{country:"**"，province:"**"，counter:"*"}

$sort

- 将输入文档排序后输出
- 例1：查询学生信息，按年龄升序

```
b.stu.aggregate({$sort:{age:1}})
```

- 例2：查询男生、女生人数，按人数降序

```
db.stu.aggregate(
    {$group:{_id:'$gender',counter:{$sum:1}}},
    {$sort:{counter:-1}}
)
```

$limit和$skip

$limit

- 限制聚合管道返回的文档数
- 例1：查询2条学生信息

```
db.stu.aggregate({$limit:2})
```

$skip

- 跳过指定数量的文档，并返回余下的文档
- 例2：查询从第3条开始的学生信息

```
db.stu.aggregate({$skip:2})
```

- 例3：统计男生、女生人数，按人数升序，取第二条数据

```
db.stu.aggregate(
    {$group:{_id:'$gender',counter:{$sum:1}}},
    {$sort:{counter:1}},
    {$skip:1},
    {$limit:1}
)
```

- 注意顺序：先写skip，再写limit

$unwind

将文档中的某一个数组类型字段拆分成多条， 每条包含数组中的一个值

语法：db.集合名称.aggregate({$unwind:'$字段名称'})

```
db.t2.insert({_id:1,item:'t-shirt',size:['S','M','L']})
db.t2.aggregate({$unwind:'$size'})
```

结果如下：

```
{ "_id" : 1, "item" : "t-shirt", "size" : "S" }
{ "_id" : 1, "item" : "t-shirt", "size" : "M" }
{ "_id" : 1, "item" : "t-shirt", "size" : "L" }
```

$unwind练习

数据库中有一条数据：{"username":"Alex","tags":['C#','Java','C++']}，如何获取该tag列表的长度?

$unwind

属性 值为false表示丢弃属性值为空的文档
属性preserveNullAndEmptyArrays值为true表示保留属性值为空的文档

用法:

```
db.inventory.aggregate({
    $unwind:{
        path:'$字段名称',
        preserveNullAndEmptyArrays:<boolean> #防止数据丢失
    }
})
```

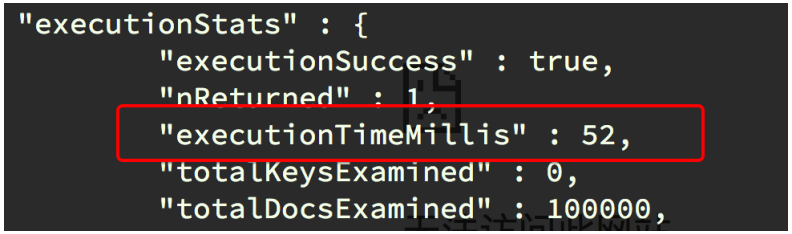
创建索引

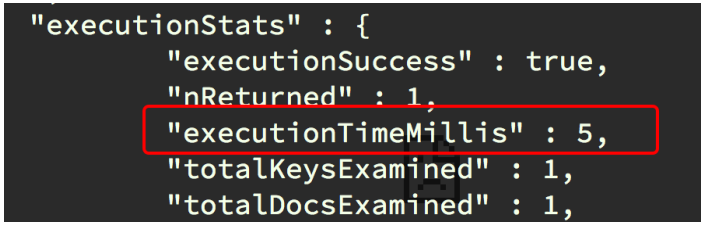
索引：以提升查询速度

测试：插入10万条数据到数据库中
for(i=0;i<100000;i++){db.t12.insert({name:'test'+i,age:i})}

db.t1.find({name:'test10000'})
db.t1.find({name:'test10000'}).explain('executionStats')

建立索引之后对比：
语法：db.集合.ensureIndex({属性:1})，1表示升序， -1表示降序
具体操作：db.t1.ensureIndex({name:1})
db.t1.find({name:'test10000'}).explain('executionStats')





索引

- 在默认情况下创建的索引均不是唯一索引。
- 创建唯一索引:
  db.t1.ensureIndex({"name":1},{"unique":true})
- 创建唯一索引并消除重复:

db.t1.ensureIndex({"name":1},{"unique":true,"dropDups":true})

- 建立联合索引(什么时候需要联合索引):
  db.t1.ensureIndex({name:1,age:1})
- 查看当前集合的所有索引:
  db.t1.getIndexes()
- 删除索引:
  db.t1.dropIndex('索引名称')

数据的备份和恢复

备份的语法：

mongodump -h dbhost -d dbname -o dbdirectory

-h： 服务器地址， 也可以指定端口号

-d： 需要备份的数据库名称

-o： 备份的数据存放位置， 此目录中存放着备份出来的数据

mongodump -h 192.168.196.128:27017 -d test1 -o ~/Desktop/test1bak

数据的恢复

恢复语法：

mongorestore -h dbhost -d dbname --dir dbdirectory

-h： 服务器地址

-d： 需要恢复的数据库实例

--dir： 备份数据所在位置

mongorestore -h 192.168.196.128:27017 -d test2 --dir ~/Desktop/test1bak/test1

动手

尝试将我电脑中的douban.tv1中的数据恢复到自己的电脑中，具体如何操作?

完成上述操作后完成以下问题:

1.获取每条数据中的title，count(所有评分人数),rate(评分),country(国家)的这些字段
2.获取上述结果中的不同国家电视剧的数据量
3.获取上述结果中分数大于8分的不同国家电视剧的数据量

实例化和插入

```
from pymongo import MongoClient

class TestMonogo:
    def __init__(self):
        client = MongoClient(host="127.0.0.1",port=27017)
        self.collection = client["test"]["t1"]  —>使用方括号的方式选择数据库和集合

    def test_insert(self):
        #insert接收字典，返回objectId
        ret = self.collection.insert({"name":"test10010","age":33})
        print(ret)

    def test_insert_many(self):
        item_list = [{"name":"test1000{}".format(i)} for i in range(10)]
        #insert_many接收一个列表，列表中为所有需要插入的字典
        t = self.collection.insert_many(item_list)
        #t.inserted_ids为所有插入的id
        for i in t.inserted_ids:
            print(i)
```

插入和更新

```
def try_find_one(self):
    #find_one查找并且返回一个结果,接收一个字典形式的条件
    t = self.collection.find_one({"name":"test10005"})
    print(t)

def try_find_many(self):
    #find返回所有满足条件的结果，如果条件为空，则返回数据库的所有
    t = self.collection.find({"name":"test10005"})
    #结果是一个Cursor游标对象，是一个可迭代对象，可以类似读文件的指针，
    for i in t:
        print(i)
    for i in t: #此时t中没有内容
        print(i)

def try_update_one(self):
    #update_one更新一条数据
    self.collection.update_one({"name":"test10005"},{"$set":{"name":"new_test10005"}})


def try_update_many(self):
    # update_one更新全部数据
    self.collection.update_many({"name":"test10005"},{"$set":{"name":"new_test10005"}})
```

删除

```
def try_delete_one(self):
    #delete_one删除一条数据
    self.collection.delete_one({"name":"test10010"})

def try_delete_many(self):
    #delete_may删除所有满足条件的数据
    self.collection.delete_many({"name":"test10010"})
```

动手一

1. 统计t1中所有的name的出现的次数
2. 统计t1中所有的name的出现的次数中次数大于4的name
3. 统计t1中所有的name的出现的次数中次数大于4的次数（只显示次数）

动手二

1. 使用python向集合t3中插入1000条文档，文档的属性包括_id、name
   - _id的值为0、1、2、3...999
   - name的值为'py0'、'py1'...

2. 查询显示出_id为100的整倍数的文档，如100、200、300...，并将name输出

Thank You!

改变中国 IT 教育，我们正在行动