Table of Contents

My Awesome Book

This file file serves as your book's preface, a great place to describe your book's content and ideas.

NoSQL简介

NoSQL=Not Only SQL，泛指非关系型的数据库。随着互联网web2.0网站的兴起，传统的关系数据库在应付web2.0网站，特别是超大规模和高并发的SNS类型的web2.0纯动态网站已经显得力不从心，暴露了很多难以克服的问题，而非关系型的数据库则由于其本身的特点得到了非常迅速的发展。

NoSQL数据库的产生就是为了解决大规模数据集合多重数据种类带来的挑战，尤其是大数据应用难题。

发展现状

NoSQL数据库现在主要有四大分类：

分类	举例	典型应用场景	数据模型	优点	缺点
key-value存储	Redis,Oracle BDB,	内容缓存，主要用于处理大量数据的高访问负载，也用于一些日志系统等等	Key指向Value的键值对,通常用hash table来实现	查找速度快	数据无结构化,通常只被当作字符串或者二进制数据
列式	HBase Cassandra Riak	分布式的	以列簇式存储，	查找速度快，可扩展性强，	功能相对

数据库	HBase,Cassandra,Riak	分布式的文件系统	将同一列数据存在一起	性强，更容易进行分布式扩展	功能相对局限
文档型数据库	CouchDB, MongoDb	Web应用（与Key-Value类似，Value是结构化的，不同的是数据库能够了解Value的内容）	Key-Value对应的键值对，Value为结构化数据	数据结构要求不严格，表结构可变，不需要关系数据库一样需要预先定义表结构	查询性能不高，而且缺乏统一的查询语法
图结构数据库	Neo4J, InfoGrid, Infinite Graph	社交网络，推荐系统等。专注于构建关系图谱	图结构	利用图结构相关算法。比如最短路径寻址，N度关系查找等	很多时候需要对整个图做计算才能得出需要的信息，而且这种结构不太好做分布式的集群方案

NoSQL特点

1. 可以处理海量数据
2. 可以运行在便宜的PC服务器集群上
3. 击碎了性能瓶颈(省去sql转换时间)
4. 没有过多的操作
5. 支持者源于社区

初识MongoDB

MongoDB是一个介于关系型数据库和非关系型数据库之间的产品,是非关系型数据库中功能最丰富,最像关系型数据库的。

文档数据库，存储的是文档(Bson->json的二进制化)

特点： 内部执行引擎为JS解释器，把文档存储成bson结构，在查询时，转换成JS对象，比可以通过熟悉的js语法来操作

Mongo和传统型数据库比，最大的不同： 传统型数据库：结构化数据，定好了表结构后，每一行的内容，必是符合表结构的，就是说列的个数，类型都一样 mongo文档型数据库: 表下的每篇文档,都可以有自己独特的结构(json对象都可以有自己独特的属性和值)

思路: 如果有电影, 影评, 影评的回复, 回复的打分 在传统型数据库中, 至少要4张表, 关联度非常复杂. 在文档数据库中,通过1篇文档,即可完成.

体现出文档型数据库的反范式化

```
{
 fiim:'天龙八部'
 comment:[
        {
            content:'王家卫的电影风格',
            reply:['支持','好']
        }
    ]
}
```

快速入门

启动mongod服务

```
./bin/mongod --dbpath /path/to/database --logpath /path/to/log --fork --port 27017

参数解释:
        --dbpath 数据存储目录
        --logpath 日志存储目录
        --port 运行端口(默认27017)
        --fork 后台进程运行
```

> ⚠mongodb非常的占磁盘空间, 刚启动后要占3-4G左右,如果你用虚拟机练习,可能空间不够,导致无法启动.可以用 `--smallfiles` 选项来启动, 将会占用较小空间400M左右.

连接

So easy~

```
Ken@localhost:~$ mongo
MongoDB shell version: 3.2.0
connecting to: test
```

连接远程mongo库

```
Ken@gaosongdeMacBook-Pro:~$ mongo 112.126.XX.XXX:27017
MongoDB shell version: 3.2.0
connecting to: 112.126.XX.XXX:27017/test
>
```

入门命令

```
show dbs  #查看当前的数据库
use databaseName #选库
show tables/collections #查看当前库下的collection

    ------------------------------------------------
    |> use shop; #Mongodb的库是隐式创建              |
    |#你可以use一个不存在的库                        |
    |#然后在该库下创建collection,即可创建库          |
    |switched to db shop                             |
    |                                                |
    |> show dbs;                                     |
    |shop           0.078GB                          |
    |> db.createCollection('user');                  |
    |{ "ok" : 1 }                                    |
    |> show collections;                             |
    |system.indexes                                  |
    |user                                            |
    ------------------------------------------------

#collection允许隐式创建
Db.collectionName.insert(document)

    ------------------------------------------------
    |> show collections;                             |
    |system.indexes                                  |
    |user                                            |
    |> db.goods.insert({_id:1,name:"goods"})         |
    |WriteResult({ "nInserted" : 1 })                |
    |> show collections;                             |
    |goods                                           |
    |system.indexes                                  |
    |user                                            |
    |> db.goods.find();                              |
    |{ "_id" : 1, "name" : "goods" }                 |
    ------------------------------------------------

#删除collection
db.collectionName.drop()
```

```
    ------------------------------------------------
    |> db.user.drop();                             |
    |true                                          |
    |> db.goods.drop();                            |
    |true                                          |
    ------------------------------------------------

#删除database
db.dropDatabase()

    ------------------------------------------------
    |> db.dropDatabase();                          |
    |{ "dropped" : "shop", "ok" : 1 }              |
    ------------------------------------------------


##查看当前用户
> show users;  #Print a list of all databases on the server.
> show roles; #Print a list of all roles, both user-defined and
built-in, for the current database.
{
    "role" : "dbAdmin",
    "db" : "shop",
    "isBuiltin" : true,
    "roles" : [ ],
    "inheritedRoles" : [ ]
}
{
    "role" : "dbOwner",
    "db" : "shop",
    "isBuiltin" : true,
    "roles" : [ ],
    "inheritedRoles" : [ ]
}
{
    "role" : "read",
    "db" : "shop",
    "isBuiltin" : true,
    "roles" : [ ],
    "inheritedRoles" : [ ]
```

```
}
{
    "role" : "readWrite",
    "db" : "shop",
    "isBuiltin" : true,
    "roles" : [ ],
    "inheritedRoles" : [ ]
}
{
    "role" : "userAdmin",
    "db" : "shop",
    "isBuiltin" : true,
    "roles" : [ ],
    "inheritedRoles" : [ ]
}
```

CRDU操作

增：insert

```
#增加单篇文档
> db.stu.insert({sn:001,name:"xiaoming"})
WriteResult({ "nInserted" : 1 })
> db.stu.find()
{ "_id" : ObjectId("5711a253c95cb3db6c1bd2b9"), "sn" : 1, "name"
 : "xiaoming" }

#增加单个文档并指定_id
> db.stu.insert({_id:2,sn:002,name:"hanmmm"})
WriteResult({ "nInserted" : 1 })
> db.stu.find()
{ "_id" : ObjectId("5711a253c95cb3db6c1bd2b9"), "sn" : 1, "name"
 : "xiaoming" }
{ "_id" : 2, "sn" : 2, "name" : "hanmmm" }

#增加多个文档
> db.stu.insert([{_id:3,sn:002,name:"xiaogang"},{sn:111,name:"jj
jj",age:9},{sn:021,age:91}])
BulkWriteResult({
    "writeErrors" : [ ],
    "writeConcernErrors" : [ ],
    "nInserted" : 3,
    "nUpserted" : 0,
    "nMatched" : 0,
    "nModified" : 0,
    "nRemoved" : 0,
    "upserted" : [ ]
})
```

删:remove

```
#语法: db.collection.remove(查询表达式, 选项);
#选项是指  {justOne:true/false},是否只删一行, 默认为false

> db.stu.find();
{ "_id" : ObjectId("5711a253c95cb3db6c1bd2b9"), "sn" : 1, "name"
 : "xiaoming" }
{ "_id" : 2, "sn" : 2, "name" : "hanmmm" }
```

```
{ "_id" : 3, "sn" : 2, "name" : "xiaogang" }
{ "_id" : ObjectId("5711a2c9c95cb3db6c1bd2ba"), "sn" : 111, "nam
e" : "jjjj", "age" : 9 }
{ "_id" : ObjectId("5711a2c9c95cb3db6c1bd2bb"), "sn" : 17, "age"
 : 91 }
> db.stu.remove({sn:2});
WriteResult({ "nRemoved" : 2 })
> db.stu.find();
{ "_id" : ObjectId("5711a253c95cb3db6c1bd2b9"), "sn" : 1, "name"
 : "xiaoming" }
{ "_id" : ObjectId("5711a2c9c95cb3db6c1bd2ba"), "sn" : 111, "nam
e" : "jjjj", "age" : 9 }
{ "_id" : ObjectId("5711a2c9c95cb3db6c1bd2bb"), "sn" : 17, "age"
 : 91 }



> db.stu.find();
{ "_id" : ObjectId("5711a253c95cb3db6c1bd2b9"), "sn" : 1, "name"
 : "xiaoming" }
{ "_id" : ObjectId("5711a2c9c95cb3db6c1bd2ba"), "sn" : 111, "nam
e" : "jjjj", "age" : 9 }
{ "_id" : ObjectId("5711a2c9c95cb3db6c1bd2bb"), "sn" : 17, "age"
 : 91 }
{ "_id" : ObjectId("5711a4c1c95cb3db6c1bd2bc"), "sn" : 1, "name"
 : "hahh" }

#只删除一行

> db.stu.remove({sn:1},true);
WriteResult({ "nRemoved" : 1 })
> db.stu.find();
{ "_id" : ObjectId("5711a2c9c95cb3db6c1bd2ba"), "sn" : 111, "nam
e" : "jjjj", "age" : 9 }
{ "_id" : ObjectId("5711a2c9c95cb3db6c1bd2bb"), "sn" : 17, "age"
 : 91 }
{ "_id" : ObjectId("5711a4c1c95cb3db6c1bd2bc"), "sn" : 1, "name"
 : "hahh" }
```

> 注意 1: 查询表达式依然是个json对象 2: 查询表达式匹配的行,将被删掉. 3: 如果不写查询表达式,collections中的所有文档将被删掉!!!

改:update

```
#语法: db.collection.update(查询表达式,新值,选项);

> db.stu.find();
{ "_id" : ObjectId("5711a4c1c95cb3db6c1bd2bc"), "sn" : 1, "name"
 : "hahh" }
#文档中的其他列也不见了,改后只有_id和name列了.即--新文档直接替换了旧文档,
而不是修改
> db.stu.update({name:"hahh"},{name:"haha"})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 }
)
> db.stu.find();
{ "_id" : ObjectId("5711a4c1c95cb3db6c1bd2bc"), "name" : "haha"
}

#如果是想修改文档的某列,可以用$set关键字

> db.stu.find();
{ "_id" : ObjectId("5711a2c9c95cb3db6c1bd2ba"), "sn" : 111, "nam
e" : "jjjj", "age" : 9 }
> db.stu.update({name:'jjjj'},{$set:{name:'kkkk'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 }
)
> db.stu.find();
{ "_id" : ObjectId("5711a2c9c95cb3db6c1bd2ba"), "sn" : 111, "nam
e" : "kkkk", "age" : 9 }

#修改时的赋值表达式
#$set  修改某列的值
#$unset 删除某个列
#$rename 重命名某个列
#$inc 增长某个列
#$setOnInsert 当upsert为true时,并且发生了insert操作时,可以补充的字段.

> db.stu.find();
```

```
{ "_id" : ObjectId("5711a2c9c95cb3db6c1bd2ba"), "sn" : 111, "nam
e" : "kkkk", "age" : 9 }
> db.stu.update({sn:111},{$set:{name:"llll"},$unset:{age:9},$inc
:{sex:1},$rename:{sn:"stn"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 }
)
> db.stu.find();
{ "_id" : ObjectId("5711a2c9c95cb3db6c1bd2ba"), "name" : "llll",
 "sex" : 1, "stn" : 111 }



#Option的作用:
#{upsert:true/false,multi:true/false}
#Upsert---是指没有匹配的行,则直接插入该行.(和mysql中的replace一样)

#例:db.stu.update({name:'wuyong'},{$set:{name:'junshiwuyong'}},{
upsert:true});
#如果有name=’wuyong’的文档,将被修改
#如果没有,将添加此新文档

> db.stu.update({sn:1024},{$set:{name:"caoliu"},$setOnInsert:{si
te:"1024.com"}},{upsert:1});
WriteResult({
    "nMatched" : 0,
    "nUpserted" : 1,
    "nModified" : 0,
    "_id" : ObjectId("5711aa9300d528448f3d8033")
})
> db.stu.find();
{ "_id" : ObjectId("5711aa9300d528448f3d8033"), "sn" : 1024, "na
me" : "caoliu", "site" : "1024.com" }



#例:
#db.news.update({_id:99},{x:123,y:234},{upsert:true});
#没有_id=99的文档被修改,因此直接插入该文档

#multi: 是指修改多行(即使查询表达式命中多行,默认也只改1行,如果想改多行,可
以用此选项)
```

```
#例:
#db.news.update({age:21},{$set:{age:22}},{multi:true});
#则把news中所有age=21的文档,都修改
```

查:find

```
#语法: db.collection.find(查询表达式,查询的列);
#Db.collections.find(表达式,{列1:1,列2:1});

#例1:db.stu.find()
#查询所有文档 所有内容

> db.stu.find();
{ "_id" : ObjectId("5711a2c9c95cb3db6c1bd2ba"), "name" : "llll",
 "sex" : 1, "stn" : 111 }
{ "_id" : ObjectId("5711a2c9c95cb3db6c1bd2bb"), "sn" : 17, "age"
 : 91 }
{ "_id" : ObjectId("5711a4c1c95cb3db6c1bd2bc"), "name" : "haha"
}
{ "_id" : ObjectId("5711aa9300d528448f3d8033"), "sn" : 1024, "na
me" : "caoliu", "site" : "1024.com" }

#例2: db.stu.find({},{gendre:1})
#查询所有文档,的gender属性 (_id属性默认总是查出来)

> db.stu.find({},{name:1})
{ "_id" : ObjectId("5711a2c9c95cb3db6c1bd2ba"), "name" : "llll"
}
{ "_id" : ObjectId("5711a2c9c95cb3db6c1bd2bb") }
{ "_id" : ObjectId("5711a4c1c95cb3db6c1bd2bc"), "name" : "haha"
}
{ "_id" : ObjectId("5711aa9300d528448f3d8033"), "name" : "caoliu
" }

#例3: db.stu.find({},{gender:1, _id:0})
#查询所有文档的gender属性,且不查询_id属性
```

```
> db.stu.find({},{name:1,_id:0})
{ "name" : "llll" }
{  }
{ "name" : "haha" }
{ "name" : "caoliu" }

#例4: db.stu.find({gender:’male’},{name:1,_id:0});
#查询所有gender属性值为male的文档中的name属性

> db.stu.find({sn:1024},{name:1,_id:0})
{ "name" : "caoliu" }
```

参考文献

- NoSQl百度百科
- 四大类NoSQL数据库