大数据库系统

第4章 文档数据库MongoDB

第4章 文档数据库MongoDB

- 本章内容

第4章 文档数据库MongoDB

4.4 MongoDB常用命令介绍

4.5 集合文档基本操作

4.5.1 插入

4.5.2 查找

4.5.4 更新

4.5.5 删除

4.1.1 MongoDB的起源

- 起源

  2007年10月，MongoDB由10gen团队开发，2009年2月首度推出，现在依然有10gen商业支持，持续性有保障

- MongoDB简介

  ① MongoDB 是由C++语言编写的，是一个基于分布式文件存储的开源数据库系统

  ② 旨在为WEB应用提供可扩展的高性能数据存储解决方案（互联网企业）

4.1.2 MongoDB的特点

- MongoDB特点

1、提供了一个面向文档存储，操作起来比较简单和容易

- 关系型数据库操作较为繁琐
- 键值数据库没有办法进行复杂的查询
- ✓ 使用高效的二进制数据存储，可以存储大型对象（如视频）

4.1.2 MongoDB的特点

- MongoDB特点

2、支持丰富的查询表达式

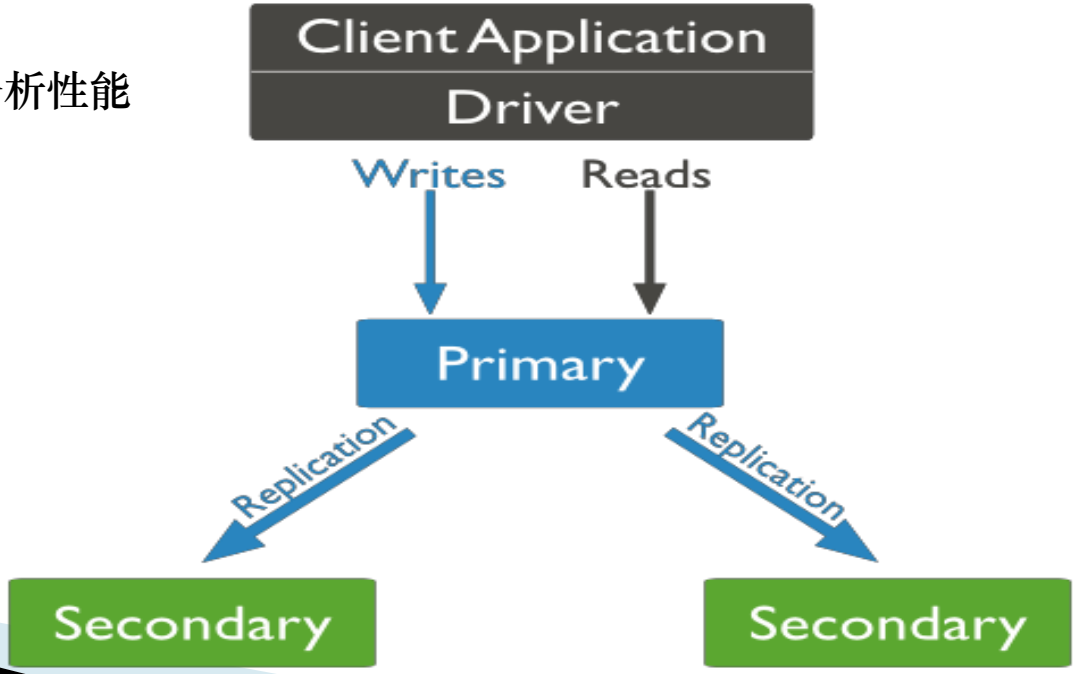
- ✓ 语法类似于面向对象的查询语言（java）
- ✓ 可以实现类似关系数据库单表查询的的大多数功能
  - MongoDB 的查询优化器会自动分析查询语句，然后生成一个高效的查询集合
  - MongoDB可以针对任何属性建立索引，更快排序
- ✓ MongoDB支持Map/Reduce，实现了关系型数据库中的group by操作

4.1.2 MongoDB的特点

◆ MongoDB特点

3、复制及自动故障转移

- ✓ 主节点的所有对数据的操作都会同步到从节点
- ✓ 主节点发生故障之后，从节点可以升级为主节点，也可以通过从节点对故障主节点进行数据恢复
- ✓ 包含一个监视工具用于分析性能



4.1.2 MongoDB的特点

- MongoDB特点

4、自动分片以支持云级别的伸缩性

- ✓ 水平扩展数据库集群，动态添加片（服务器）
- ✓ 保证存储的负载均衡

5、可以通过网络访问

- ✓ 可以通过网络远程访问MongoDB数据库

4.1.2 MongoDB的特点

- MongoDB特点

6、支持多种编程语言驱动：

- ✓ PYTHON
- ✓ PHP
- ✓ JAVA
- ✓ C、C++、C#等

4.1.2 MongoDB的特点

- MongoDB特点

7、MongoDB 是一个面向集合且模式自由的文档类型数据库

A. 面向集合

- 面向集合是指数据被分组存在在数据集中，被称为一个集合
- 集合类似于关系型数据库的表table
- 集合不需要定义任何模式，集合存储文档

4.1.2 MongoDB的特点

- MongoDB特点

7、MongoDB 是一个面向集合且模式自由的文档类型数据库

B. 模式自由

- 存储在 MongoDB 数据库中的数据，不需要知道它是什么结构（可以是任何的文档）

C. 文档型

- 键值对的集合，键是字符串，值可以是数据类型集合中的任意类型
- 数据格式称为 BSON （Binary Serialized Document Notation）

4.1.2 MongoDB的特点

通过下图实例，我们也可以更直观的的了解MongoDB中的一些概念：

id	user_name	email	age	city
1	Mark Hanks	mark@abc.com	25	Los Angeles
2	Richard Peter	richard@abc.com	31	Dallas

```
{
        "_id": ObjectId("5146bb52d8524270060001f3"),
        "age": 25,
        "city": "Los Angeles",
        "email": "mark@abc.com",
       "user_name": "Mark Hanks "
}
{       "_id": ObjectId("5146bb52d8524270060001f2"),
        "age": 31,
        "city": "Dallas",
        "email": "richard@abc.com",
        "user_name": "Richard Peter "
}
```

4.1.3 MongoDB适用场景

- MongoDB 的主要目标
  - 在键/值存储方式（提供了高性能和高度伸缩性）和传统的 RDBMS 系统（具有丰富的功能）之间架起一座桥梁，它集两者的优势于一身
  - 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的，最像关系型数据库

4.1.3 MongoDB适用场景

- MongoDB 适用场景
  - ✓高伸缩性的场景

    MongoDB 非常适合实时的插入，更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性
  - ✓缓存

    由于性能很高，MongoDB也适合作为信息基础设施的缓存层
  - ✓海量数据存储

    MongoDB非常适合由数十或数百台服务器组成的数据库， MongoDB的路线图中已经包含对MapReduce引擎的内置支持。
  - ✓用于对象及JSON 数据的查询

    MongoDB的BSON数据格式非常适合文档化格式的存储及查询

4.1.3 MongoDB适用场景

◆ 具体应用

- 游戏场景

使用MongoDB存储游戏用户信息，用户的装备、积分等直接以内嵌文档的形式存储，方便查询、更新

- 电商场景

使用MongoDB存储商品图片等信息，订单状态在运送过程中会不断更新，以MongoDB内嵌数组的形式来存储，一次查询就能将订单所有的变更读取出来。

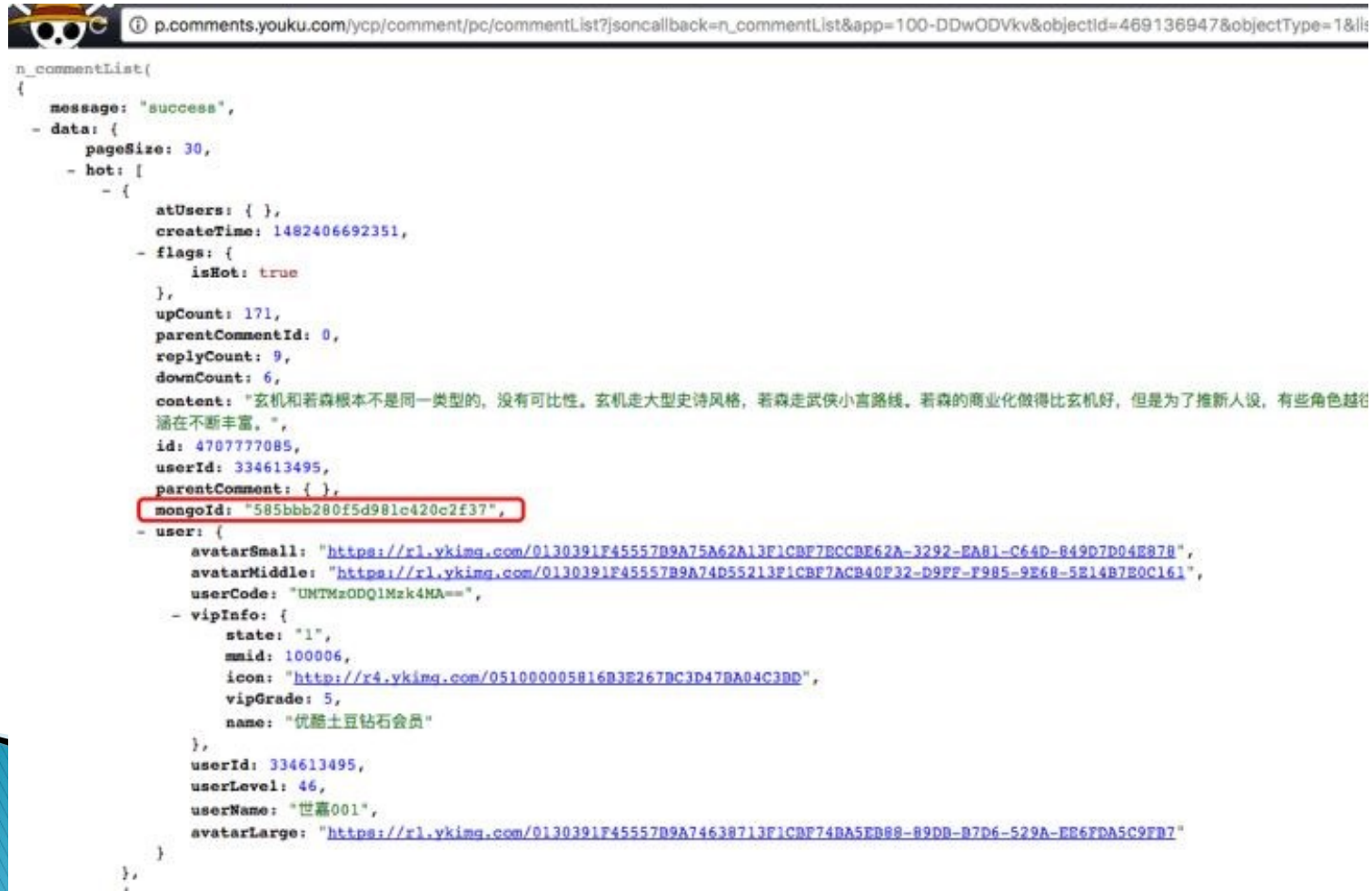
- 社交场景

使用MongoDB存储存储用户信息，以及用户发表的朋友圈信息，通过地理位置索引实现附近的人、地点等功能。

4.1.3 MongoDB适用场景

- 企业应用

优酷视频评论使用MongoDB存储



4.1.3 MongoDB适用场景

◆ 企业应用

饿了么

- 将送快递骑手、快递商家的信息（包含位置信息）存储在 MongoDB，然后通过 MongoDB 的地理位置查询，这样很方便的实现了查找附近的商家、骑手等功能，使得快递骑手能就近接单

其他国内企业应用

- 小红书的核心系统，高德地图的app展示，千寻位置的日志收集分析，阴阳师的数据库，360的移动搜索等

4.1.3 MongoDB适用场景

- 不适用场景

高度事务性的系统

- 例如，银行或会计系统
- 传统的关系型数据库目前还是更适用于需要大量原子性复杂事务的应用程序

传统的商业智能应用

- 对于此类应用，数据仓库可能是更合适的选择
- 复杂的跨文档（表）级联查询

4.2 MongoDB的安装

- 安装MongoDB

1. Window平台安装MongoDB

MongoDB提供了可用于32位和64位系统的预编译二进制包，可在MongoDB官网下载安装

下载地址：http://www.mongodb.org/downloads

注意：在MongoDB2.2版本后已经不再支持Window XP系统

4.2 MongoDB的安装

- 安装MongoDB

Linux平台安装MongoDB

安装方法一：MongoDB提供了Linux平台于32位和64位的安装包，可在MongoDB官网下载安装包

下载地址：http://www.mongodb.org/downloads

安装步骤：http://dblab.xmu.edu.cn/blog/868-2/

启动MongoDB服务

在MongoDB安装目录的bin目录下执行mongod即可

4.2 MongoDB的安装

- Linux平台安装MongoDB （2.6.10）

  安装方法二：需要联网

  使用命令：

  sudo apt-get install mongodb

  可下载安装MongoDB，默认安装的版本是MongoDB 2.6.10

  如果想安装更高版本3.2.8版，可考虑按如下步骤：

4.2 MongoDB的安装

- Linux平台安装MongoDB（安装3.2.8版本）

1、输入以下命令：

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927

echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.2 multiverse" |
sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list

sudo apt-get update
```

出错见下页

2、成功后再执行：

```
sudo apt-get install -y mongodb-org --allow-unauthenticated
```

4.2 MongoDB的安装

- 在运行sudo apt-get update时出错

错误一：

Reading package lists... Done E: Problem executing scripts APT::Update::Post-Invoke-Success 'if /usr/bin/test -w /var/cache/app-info -a -e /usr/bin/appstreamcli; then appstreamcli refresh > /dev/null; fi' E: Sub-process returned an error code

解决方法

依次执行以下命令

sudo pkill -KILL appstreamcli

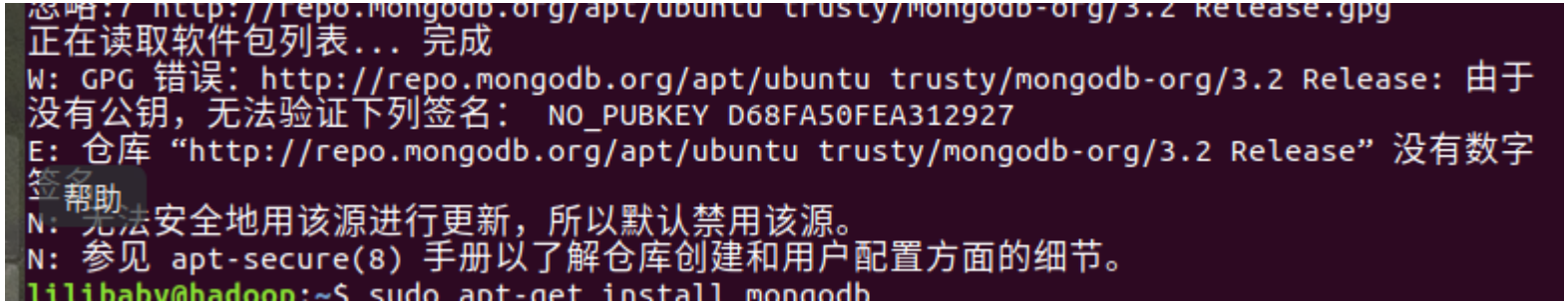
wget -P /tmp https://launchpad.net/ubuntu/+archive/primary/+files/appstream_0.9.4-1ubuntu1_amd64.deb https://launchpad.net/ubuntu/+archive/primary/+files/libappstream3_0.9.4-1ubuntu1_amd64.deb

sudo dpkg -i /tmp/appstream_0.9.4-1ubuntu1_amd64.deb /tmp/libappstream3_0.9.4-1ubuntu1_amd64.deb

执行完上述命令之后再次运行sudo apt-get update

◆ 在运行sudo apt-get update时出错

错误二：



解决方法

sudo apt-key adv –keyserver hkp://keyserver.ubuntu.com:80 –recv D68FA50FEA312927

4.2 MongoDB的安装

- 在运行sudo apt-get update时出错

错误三：

…

update completed, but some metadata was ignored due to errors.

E: 无法获得锁 /var/lib/dpkg/lock – open (11: 资源暂时不可用)

E: 无法锁定管理目录(/var/lib/dpkg/)，是否有其他进程正占用它?

…

解决方法

依次输入以下三条命令：

sudo rm /var/cache/apt/archives/lock

sudo rm /var/lib/dpkg/lock

sudo apt-get update

4.2 MongoDB的安装

- 安装完成后

  启动和关闭mongodb命令如下：

  sudo service mongodb start

  sudo service mongodb stop

4.2 MongoDB的安装

◆输入“sudo service mongodb start”启动mongodb出错

Failed to start mongod.service: Unit not found

◆解决方法

1、使用vim编辑器创建配置文件

sudo vim /etc/systemd/system/mongodb.service

2、在该配置文件中添加如下内容：

[Unit]

Description=High-performance, schema-free document-oriented database

After=network.target

[Service]

User=mongodb

ExecStart=/usr/bin/mongod --quiet --config /etc/mongod.conf

[Install]

WantedBy=multi-user.target

保存退出vim编辑器

4.2 MongoDB的安装

- 解决方法

  3、再次尝试

  sudo service mongodb start

  mongo

  如果mongo提示连不上，尝试重启linux后再试一次上述语句

  或输入以下指令

  sudo systemctl start mongodb

  sudo systemctl status mongodb

  sudo systemctl enable mongodb

  sudo service mongodb start

4.2 MongoDB的安装

- 通过apt-get方法下载安装MongoDB，系统会自动指定安装路径和默认的MongoDB数据库的存放位置
- 查看默认数据库、日志、配置文件的存放位置

1、安装完成后，在linux的命令行输入mongo进入mongoDB

```
hadoop@fisher-VirtualBox:~$ mongo
MongoDB shell version: 3.2.22
connecting to: test
Server has startup warnings:
2021-01-28T21:46:06.064+0800 I CONTROL  [initandlisten]
2021-01-28T21:46:06.064+0800 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/
mm/transparent_hugepage/defrag is 'always'.
2021-01-28T21:46:06.064+0800 I CONTROL  [initandlisten] **        We suggest set
ting it to 'never'
2021-01-28T21:46:06.064+0800 I CONTROL  [initandlisten]
>
```

2、再开一个终端，使用ps命令查看mongo进程

```
hadoop@fisher-VirtualBox:/usr/share$ ps -ef | grep mongo
mongodb  20246     1  0 14:37 ?        00:00:02 /usr/bin/mongod --config /etc/mo
ngodb.conf
```

可以看到配置文件位置/etc/mongodb.conf

4.2 MongoDB的安装

3、使用vim命令查看/etc/mongodb.conf

```
hadoop@fisher-VirtualBox:/usr/share$ vim /etc/mongodb.conf
```

```
# mongodb.conf

# Where to store the data.
dbpath=/var/lib/mongodb

#where to log
logpath=/var/log/mongodb/mongodb.log

logappend=true

bind_ip = 127.0.0.1
#port = 27017
```

从配置文件中可以看到默认的数据库路径为/var/lib/mongodb

默认的日志存放目录为/var/log/mongodb/mongodb.log

安装完成后，默认设置MongoDB服务是随Ubuntu启动自动启动的

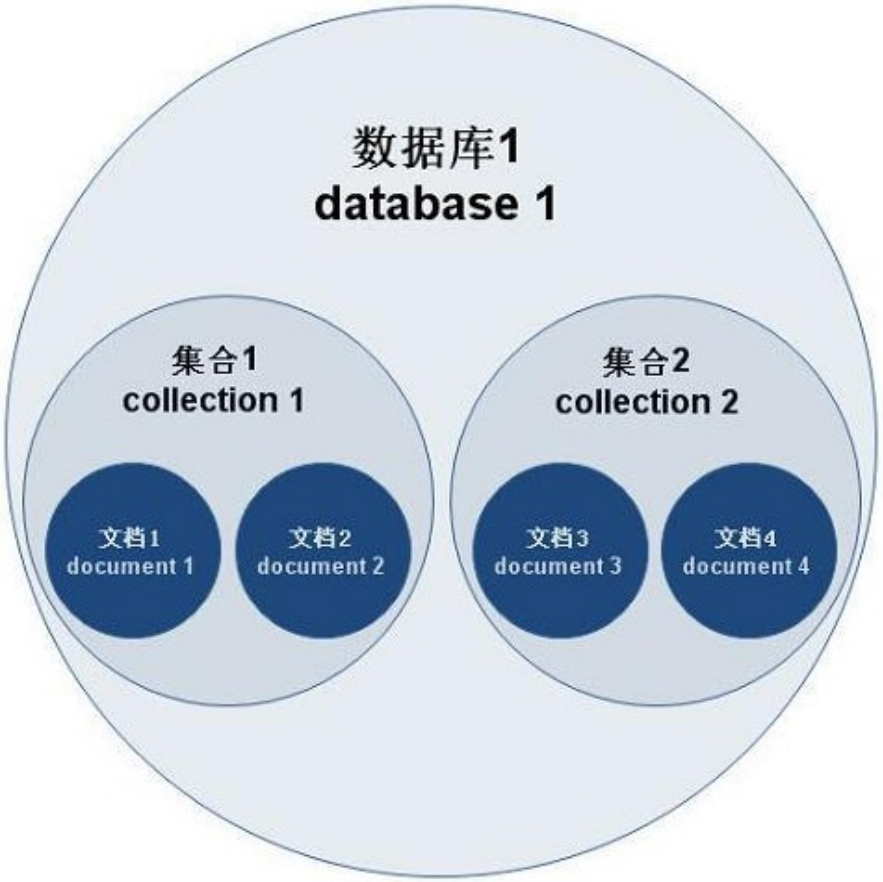
4.3 MongoDB的概念介绍

- MongoDB的概念解析——概念术语

SQL术语/概念	MongoDB术语/概念	解释/说明
database	database	数据库
table	collection	数据库表/集合
row	document	数据记录行/文档
column	field	数据字段/域
index	index	索引
table joins		表连接,MongoDB不支持
primary key	primary key	主键,MongoDB自动将_id字段设置为主键

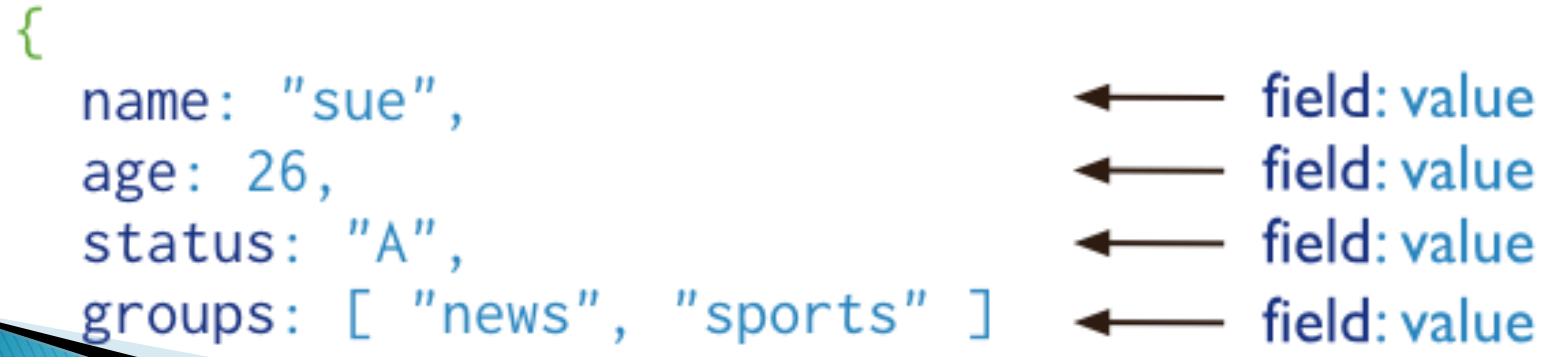
4.3 MongoDB的概念介绍

- MongoDB的概念解析
  - ✓一个MongoDB 实例可以包含一组数据库
  - ✓一个Database 可以包含一组Collection（集合）
  - ✓一个集合可以包含一组Document（文档）



4.3.1 文档

- 文档
  - 文档是 MongoDB 中数据的基本单位，类似于关系数据库中的行，多个键及其关联的值有序地放在一起就构成了文档
  - 文档的数据结构由键值(key=>value)对组成
  - 文档类似于 JSON 对象。字段值可以包含其他文档，数组及文档数组



4.3.1 文档

- 文档的注意事项

a. MongoDB不但区分数据类型，也区分大小写

{"user" : "11"}与{"user" : 11}不同

{"User" : "11"}与{"user" : 11}不同

b.文档中不允许有重复的键

{"userName" : "bbs11"，"userName" : "david"}为非法

c.文档中键/值对是有序的

{"userName" : "bbs59", "passwd" : "ddddddd", "acctAttr" : null }

{"passwd" : "ddddddd", "acctAttr" : null , "userName" : "bbs59" }

4.3.1 文档

- 文档的注意事项

d. 文档中的值不仅可以是字符串，也可以是其它数据类型（或者嵌入其它文档）

author:

pid	tid	name
1	1	Jane

blogposts:

tid	cid	title
1	1	"MyFirstPost"
1	2	"MyFirstPost"

comments:

cid	by	text
1	"Abe"	"First"
2	"Ada"	"Good post"

4.3.1 文档

- 文档的注意事项

关系数据库中的其中一条记录，在文档数据库MongoDB中的存储方式类似如下：

```
{
“id”:1,
“author”:”Jane”,
“blogposts”:{
                “tile”:”MyFirstPost”, “comment”:{
                                                “by”:”Ada”,”text”:”Good post”
                                                }
                }
}
```

4.3.1 文档

◆ 文档的注意事项

e. 键是字符串，值可以使用任意UTF-8字符

- 键的名称只能是字符串，不能是其他类型
- “.”、“$”、“_”做为保留字符，不能出现在键的名称中

f. 键不能含有\0(空字符)，空字符表示键的结尾

4.3.2 集合

- 集合


```
{"site":"www.baidu.com"}
{ “site” : “fzu.edu.cn” , “name” : “福州大学"}
{"site":"www.runoob.com","name":"菜鸟教程","num":5}
```

4.3.2 集合

- 集合命名的注意事项
  a. 集合名不能是空串"";
  b. 不能含有空字符\0;
  c. 不能以"system."开头，这是系统集合保留的前缀
  d. 集合名不能含保留字符$

4.3.3 数据库

- 数据库
  - 多个集合组成数据库
  - 磁盘上，MongoDB将不同数据库存放在不同文件中
  - MongoDB的单个实例可以容纳多个独立的数据库，每一个都有自己的集合和权限，不同的数据库也放置在不同的文件中

4.3.3 数据库

- 数据库命名规则
  a. 数据库名是UTF-8字符串，最长64个字符
  b. 不能是空字符串“”
  c. 不能含"、.、$、/、\和\0
  d. 不能与系统保留数据库同名

4.3.3 数据库

- 系统保留数据库

Admin

root数据库，添加用户到该数据库中，该用户会自动继承所有数据库权限

Local

这个数据库中的数据永远不会被复制，可以用于存储限于本地数据单台服务器的任意集合

Config

分片时，config数据库在内部使用，保存分片信息

4.4 MongoDB常用命令介绍

数据类型	描述
String	字符串。存储数据常用的数据类型。在 MongoDB 中，UTF-8 编码的字符串才是合法的。
Integer	整型数值。用于存储数值，可分为 32 位或 64 位。
Boolean	布尔值。用于存储布尔值（真/假）。
Double	双精度浮点值。用于存储浮点值。
Min/Max keys	将一个值与 BSON（二进制的 JSON）元素的最低值和最高值相对比。
Arrays	用于将数组或列表或多个值存储为一个键。
Timestamp	时间戳。记录文档修改或添加的具体时间。
Object	用于内嵌文档。
Null	用于创建空值。
Symbol	符号。该数据类型基本上等同于字符串类型，但不同的是，它一般用于采用特殊符号类型的语言。
Date	日期时间。用 UNIX 时间格式来存储当前日期或时间。你可以指定自己的日期时间：创建 Date 对象，传入年月日信息。
Object ID	对象 ID。用于创建文档的 ID。
Binary Data	二进制数据。用于存储二进制数据。
Code	代码类型。用于在文档中存储 JavaScript 代码。
Regular expression	正则表达式类型。用于存储正则表达式。

4.4 MongoDB常用命令介绍

- 常用操作命令：

输入如下命令进入MongoDB的shell命令模式：

mongo

默认连接的数据库是test数据库

```
hadoop@fisher-VirtualBox:~$ mongo
MongoDB shell version: 3.2.22
connecting to: test
Server has startup warnings:
2021-01-28T21:46:06.064+0800 I CONTROL  [initandlisten]
2021-01-28T21:46:06.064+0800 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/
mm/transparent_hugepage/defrag is 'always'.
2021-01-28T21:46:06.064+0800 I CONTROL  [initandlisten] **        We suggest set
ting it to 'never'
2021-01-28T21:46:06.064+0800 I CONTROL  [initandlisten]
>
```

4.4 MongoDB常用命令介绍

- 数据库相关操作命令：

  show dbs:显示数据库列表

  show collections：显示当前数据库中的集合（类似关系数据库中的表table）

  show users：显示所有用户

  use DBname：切换当前数据库至DBname

  db.help()：显示数据库操作命令

  db.yourCollection.help()：显示集合操作命令，yourCollection是集合名

4.4 MongoDB常用命令介绍

- 常用操作命令：

1、创建一个名为“School”的数据库

MongoDB没有创建数据库的命令，先运行use School命令

2、接着创建集合db.createCollection(‘teacher’)，可以创建一个名叫“teacher”的集合

```
> show dbs
admin   0.078GB
local   0.078GB
test    0.078GB
> use School
switched to db School
> db.createCollection('teacher')
{ "ok" : 1 }
> show dbs
School  0.078GB
admin   0.078GB
local   0.078GB
test    0.078GB
> show collections
system.indexes
teacher
>
```

4.5 集合文档基本操作

◆ 插入数据

与数据库创建类似，插入数据时也会自动创建集合

插入数据有两种方式：insert和save

```
db.student.insert({_id:1, sname: 'zhangsan', sage: 20}) #_id可选
db.student.save({_id:1, sname: 'zhangsan', sage: 22}) #_id可选
```

- 两种方式插入的数据中_id字段均可不写，会自动生成一个唯一的_id来标识本条数据。
- insert和save不同之处在于：在手动插入_id字段时，如果_id已经存在，insert不做操作，save做更新操作；如果不加_id字段，两者作用相同都是插入数据
- 键名可以不加引号，值如果是字符串必须加引号，数字不加引号为整型，加引号为字符串

例：

1、使用insert插入数据，注意这条数据的的id为1

```
> db.student.insert({_id:1, sname: 'zhangsan', sage: 20})
WriteResult({ "nInserted" : 1 })
> db.student.find()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 20 }
```

insert：插入成功

2、使用save命令同样插入_id=1的数据

```
> db.student.save({_id:1, sname: 'zhangsan', sage: 22})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.student.find()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
```

save：id相同，更新数据

可以看到sage由20更新成了22，成功更新

3、使用insert插入数据同样再次插入_id=1的数据

```
> db.student.insert({_id:1, sname: 'zhangsan', sage: 25})
WriteResult({
        "nInserted" : 0,
        "writeError" : {
                "code" : 11000,
                "errmsg" : "insertDocument :: caused by :: 11000 E11000 du
plicate key error index: School.student.$_id_  dup key: { : 1.0 }"
        }
})
> db.student.find()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
```

insert：_id相同，插入失败，不做操作

insert报错，再次查询时sage依然为22

4.5 集合文档基本操作

- 插入数据
  - 添加的数据其结构是松散的，列属性均不固定，根据添加的数据为准
  - 可以一次性插入多条文档，这时可以考虑先定义数据再插入，就可以一次性插入多条数据
  - 文档大小不能超过4M
  - 批量插入最大16M

例：批量插入文档

1、定义s，s的内容为所有文档数据

```
> s = [{sname:'lisi',sage:20},{sname:'wangwu',sage:20},{sname:'chenliu',sage:20}]
[
        {
                "sname" : "lisi",
                "sage" : 20
        },
        {
                "sname" : "wangwu",
                "sage" : 20
        },
        {
                "sname" : "chenliu",
                "sage" : 20
        }
]
```

2、向student集合批量插入文档

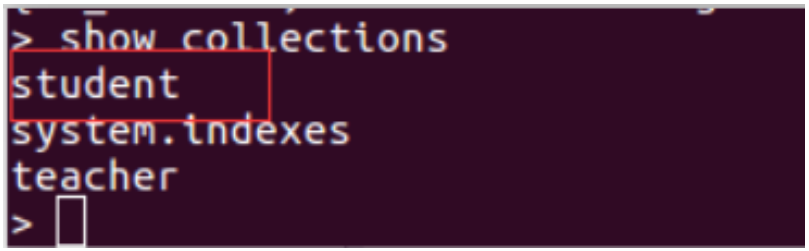
```
> db.student.insert(s)
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 3,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
> db.student.find()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("577b3a1dacae8736eab65abf"), "sname" : "lisi", "sage" : 20 }
{ "_id" : ObjectId("577b3a1dacae8736eab65ac0"), "sname" : "wangwu", "sage" : 20 }
{ "_id" : ObjectId("577b3a1dacae8736eab65ac1"), "sname" : "chenliu", "sage" : 20 }
>
```

可以看到成功批量插入数据

4.5 集合文档基本操作

- 插入数据

运行完以上例子，student 已自动创建，这也说明 MongoDB不需要预先定义 collection ，在第一次插入数据后，collection 会自动的创建



4.5 集合文档基本操作

- 查找数据

Find命令

格式：db.youCollection.find(criteria, filterDisplay)

criteria ： 查询条件，可选

filterDisplay：筛选显示部分域，

- 如显示指定列数据，可选（当选择时，第一个参数不可省略，若查询条件为空，可用{}做占位符）

youCollection：集合名

1、db.student.find()

- 查询所有记录，相当于：select * from student

```
> db.student.find()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 20 }
{ "_id" : ObjectId("60127104939843297efc4bc6"), "sname" : "wangwu", "sage" : 20
}
{ "_id" : ObjectId("6012710f939843297efc4bc7"), "sname" : "chenliu", "sage" : 20
 }
>
```

2、db.student.find({sname: 'lisi'})

- 查询sname=‘lisi’的数据，相当于： select * from student where sname='lisi'

```
> db.student.find({sname: 'lisi'})
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 20 }
>
```

3、db.student.find({},{sname:1, sage:1})

- 查询指定列sname、sage数据。1表示返回sname列，默认_id字段也是返回的，可以添加_id:0（意为不返回_id）写成{sname: 1, sage: 1,_id:0}，就不会返回默认的_id字段了
- 相当于：select sname,sage from student

```
> db.student.find({},{sname:1, sage:1})
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 20 }
{ "_id" : ObjectId("60127104939843297efc4bc6"), "sname" : "wangwu", "sage" : 20
}
{ "_id" : ObjectId("6012710f939843297efc4bc7"), "sname" : "chenliu", "sage" : 20
 }
>
```

4、db.student.find({sname: 'zhangsan', sage: 22})

- and 与条件查询
- 相当于：select * from student where sname = 'zhangsan' and sage = 22

```
> db.student.find({sname: 'zhangsan', sage: 22})
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
>
```

5、db.student.find({$or: [{sage: 22}, {sage: 25}]})

- or 条件查询。
- 相当于：select * from student where sage = 22 or sage = 25

```
> db.student.save({sname:'zhaoqi',sage:25})
WriteResult({ "nInserted" : 1 })
> db.student.find({$or: [{sage: 22}, {sage: 25}]})
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi", "sage" : 25
}
```

6、db.student.find({},{sname:1, sage:1, _id:0})

```
> db.student.find({},{sname:1, sage:1, _id:0})
{ "sname" : "zhangsan", "sage" : 22 }
{ "sname" : "lisi", "sage" : 20 }
{ "sname" : "wangwu", "sage" : 20 }
{ "sname" : "chenliu", "sage" : 20 }
>
```

7、db.student.find({sname: 'zhangsan', sage: 22}, {sname:1})

```
> db.student.find({sname: 'zhangsan', sage: 22}, {sname:1})
{ "_id" : 1, "sname" : "zhangsan" }
```

8、db.student.find({$or: [{sage: 22}, {sage: 25}]},{sage:0})

```
> db.student.find({$or: [{sage: 22}, {sage: 25}]},{sage:0})
{ "_id" : 1, "sname" : "zhangsan" }
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi" }
```

4.5 集合文档基本操作

◆ 查找数据

Find命令

比较操作符lt、lte、gt、gte、ne分别对应<、<=、>、>=、!=

1、db.student.find({sage:{"$gt":21,"$lt":26}})

- 相当于select * from student where sage >21 and sage <26;

```
> db.student.find({sage:{"$gt":21,"$lt":26}})
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi", "sage" : 25
}
```

2、db.student.find({sage:{"$gte":20,"$lte":22}})

- 相当于select * from student where sage >=20 and sage <=22;

```
> db.student.find({sage:{"$gte":20,"$lte":22}})
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 20 }
{ "_id" : ObjectId("60127104939843297efc4bc6"), "sname" : "wangwu", "sage" : 20
}
{ "_id" : ObjectId("6012710f939843297efc4bc7"), "sname" : "chenliu", "sage" : 20
 }
```

4.5 集合文档基本操作

◆ 查找数据

Find命令

3、db.student.find({sage:{"$ne":20}})

- 相当于select * from student where sage !=20;

```
> db.student.find({sage:{"$ne":20}})
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi", "sage" : 25
}
```

4、 db.student.find({sname:'zhangsan',sage:{'$ne':20}},{sname:1})

```
> db.student.find({sname:'zhangsan',sage:{'$ne':20}},{sname:1})
{ "_id" : 1, "sname" : "zhangsan" }
>
```

4.5 集合文档基本操作

◆ 查找数据

Find命令

in，not in ($in, $nin)

1、db.student.find({sage:{"$in":[20,21,22]}})

相当于select * from student where sage in (20,21,22)

```
> db.student.find({sage:{"$in":[20,21,22]}})
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 20 }
{ "_id" : ObjectId("60127104939843297efc4bc6"), "sname" : "wangwu", "sage" : 20
}
{ "_id" : ObjectId("6012710f939843297efc4bc7"), "sname" : "chenliu", "sage" : 20
 }
```

2、db.student.find({sage:{"$nin":[20,21,22]}})

相当于select * from student where sage not in (20,21,22)

```
> db.student.find({sage:{"$nin":[20,21,22]}})
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi", "sage" : 25
}
```

3、 db.student.find({sname:'lisi',sage:{"$in":[20,21,22]}},{sname:1,_id:0})

```
> db.student.find({sname:'lisi',sage:{"$in":[20,21,22]}},{sname:1,_id:0})
{ "sname" : "lisi" }
```

4.5 集合文档基本操作

- 查找数据

  排序

  在 MongoDB 中使用find中的sort() 方法对数据进行排序，sort() 方法可以通过参数指定排序的字段，并使用 1 和 -1 来指定排序的方式，其中 1 为升序排列，而 -1 是用于降序排列

4.5 集合文档基本操作

例:

1、db.student.find({sage:{"$gt":21,"$lt":26}}). sort({sage:1})

相当于select * from student where sage >21 and sage <26 asc;

```
> db.student.find({sage:{"$gt":21,"$lt":26}}). sort({sage:1})
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi", "sage" : 25
}
```

2、db.student.find({sage:{"$gt":21,"$lt":26}}). sort({sage:-1})

相当于select * from student where sage >21 and sage <26 desc;

```
> db.student.find({sage:{"$gt":21,"$lt":26}}). sort({sage:-1})
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi", "sage" : 25
}
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
```

4.5 集合文档基本操作

- 查找数据

Find命令

排序limit

如果需要读取指定数量的数据记录，可以使用find的Limit方法，limit()

接受一个数字参数，该参数指定从MongoDB中读取的记录条数

db.student.find({sage:{"$gt":19,"$lt":26}}). limit(2)

```
> db.student.find({sage:{"$gt":19,"$lt":26}}). limit(2)
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 20 }
>
```

4.5 集合文档基本操作

- 查找数据

Find命令

count统计

db.student.find({sage:{"$gt":19,"$lt":26}}). count()

相当于select count(*) from student where sage >19 and sage <26

```
> db.student.find({sage:{"$gt":19,"$lt":26}}). count()
5
```

4.5 集合文档基本操作

- 查找数据

Find命令

pretty()

格式化输出

db.student.find({sage:{"$in":[20,21,22]}}). pretty()

```
> db.student.find({sage:{"$in":[20,21,22]}}). pretty()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{
        "_id" : ObjectId("601270fa939843297efc4bc5"),
        "sname" : "lisi",
        "sage" : 20
}
{
        "_id" : ObjectId("60127104939843297efc4bc6"),
        "sname" : "wangwu",
        "sage" : 20
}
{
        "_id" : ObjectId("6012710f939843297efc4bc7"),
        "sname" : "chenliu",
        "sage" : 20
}
```

◆ 修改数据：update命令

db.youCollection.update(criteria, objNew, upsert, multi)

- criteria: update的查询条件，类似sql update查询内where后面的

- objNew : update的对象和一些更新的操作符（如$set，修改某个字段的值）

- upsert : 如果不存在update的记录，是否执行objNew，true为执行，默认是false，不执行

- multi: mongodb默认是false,只更新找到的第一条记录；如果这个参数为true,就把按条件查出来多条记录全部更新。默认false，只修改匹配到的第一条数据

其中criteria和objNew是必选参数，upsert和multi可选参数

4.5 集合文档基本操作

- 字段更新操作符： $set

$set用来指定一个键的值

1、db.student.update({sname: 'lisi'}, {$set: {sage: 30}}, false, true)

- 相当于：update student set sage =30 where sname = 'lisi';

```
> db.student.update({sname: 'lisi'}, {$set: {sage: 30}}, false, true)
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.student.find()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 30 }
{ "_id" : ObjectId("60127104939843297efc4bc6"), "sname" : "wangwu", "sage" : 20
}
{ "_id" : ObjectId("6012710f939843297efc4bc7"), "sname" : "chenliu", "sage" : 20
 }
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi", "sage" : 25
}
```

2、db.student.update({sname: 'Tom'}, {$set: {sage: 30}})

- 什么都不做，没有Tom，因为upsert=false

```
> db.student.update({sname: 'Tom'}, {$set: {sage: 30}})
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
```

3、db.student.update({sname: 'Tom'}, {$set: {sage: 30}},true,false)

- 因为upsert=true，会添加Tom

```
> db.student.update({sname: 'Tom'}, {$set: {sage: 30}},true,false)
WriteResult({
        "nMatched" : 0,
        "nUpserted" : 1,
        "nModified" : 0,
        "_id" : ObjectId("60128c45422ca12f6d93b595")
})
> db.student.find()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 30 }
{ "_id" : ObjectId("60127104939843297efc4bc6"), "sname" : "wangwu", "sage" : 20
}
{ "_id" : ObjectId("6012710f939843297efc4bc7"), "sname" : "chenliu", "sage" : 20
 }
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi", "sage" : 25
}
{ "_id" : ObjectId("60128c45422ca12f6d93b595"), "sname" : "Tom", "sage" : 30 }
```

4.5 集合文档基本操作

- 字段更新操作符

$unset

从文档中移除指定的键

db.student.update({sname: 'Tom'}, {$unset: {sage: 1}})

删除Tom的sage的键

```
> db.student.update({sname: 'Tom'}, {$unset: {sage: 1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.student.find()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 30 }
{ "_id" : ObjectId("60127104939843297efc4bc6"), "sname" : "wangwu", "sage" : 20
}
{ "_id" : ObjectId("6012710f939843297efc4bc7"), "sname" : "chenliu", "sage" : 20
 }
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi", "sage" : 25
}
{ "_id" : ObjectId("60128c45422ca12f6d93b595"), "sname" : "Tom" }
```

4.5 集合文档基本操作

- 字段更新操作符

$inc

增加已有键的值，或者在键不存在时创建一个键。$inc只能用于整数、长整数或双精度浮点数。用在其他类型的数据上就会导致操作失败

还原Tom的sage域为

1、db.student.update({sname: 'Tom'}, {$inc: {sage: -5}},true,false)

Tom的sage键的值减少5

```
{ "_id" : ObjectId("60128e5b422ca12f6d93b597"), "sname" : "Tom", "sage" : 30 }
> db.student.update({sname: 'Tom'}, {$inc: {sage: -5}},true,false)
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.student.find()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 30 }
{ "_id" : ObjectId("60127104939843297efc4bc6"), "sname" : "wangwu", "sage" : 20
}
{ "_id" : ObjectId("6012710f939843297efc4bc7"), "sname" : "chenliu", "sage" : 20
 }
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi", "sage" : 25
}
{ "_id" : ObjectId("60128e5b422ca12f6d93b597"), "sname" : "Tom", "sage" : 25 }
```

2、db.student.update({sname: 'aaa'}, {$inc: {sage: 5}},false,false)

操作无效，sname没有aaa，并且upsert=false

```
> db.student.update({sname: 'aaa'}, {$inc: {sage: 5}},false,false)
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
```

3、db.student.update({sname: 'aaa'}, {$inc: {sage: 5}},true,false)

操作生效， upsert=true，由于sname没有aaa，会增加sname='aaa'和sage=5

```
> db.student.update({sname: 'aaa'}, {$inc: {sage: 5}},true,false)
WriteResult({
        "nMatched" : 0,
        "nUpserted" : 1,
        "nModified" : 0,
        "_id" : ObjectId("60129248422ca12f6d93b598")
})
> db.student.find()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 30 }
{ "_id" : ObjectId("60127104939843297efc4bc6"), "sname" : "wangwu", "sage" : 20
}
{ "_id" : ObjectId("6012710f939843297efc4bc7"), "sname" : "chenliu", "sage" : 20
 }
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi", "sage" : 25
}
{ "_id" : ObjectId("60128e5b422ca12f6d93b597"), "sname" : "Tom", "sage" : 25 }
{ "_id" : ObjectId("60129248422ca12f6d93b598"), "sname" : "aaa", "sage" : 5 }
```

4.5 集合文档基本操作

4、找出所有sage为20的文档，将其年龄+5

db.student.update({sage: 20}, {$inc : {sage: 5}},false,true)

```
> db.student.find()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 30 }
{ "_id" : ObjectId("60127104939843297efc4bc6"), "sname" : "wangwu", "sage" : 20
}
{ "_id" : ObjectId("6012710f939843297efc4bc7"), "sname" : "chenliu", "sage" : 20
 }
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi", "sage" : 25
}
{ "_id" : ObjectId("60128e5b422ca12f6d93b597"), "sage" : "Tom" }
{ "_id" : ObjectId("601294f7422ca12f6d93b599"), "sname" : "aaa", "age" : 5 }
{ "_id" : ObjectId("60129689422ca12f6d93b59a"), "sage" : "lucy" }
```

```
> db.student.update({sage: 20}, {$inc : {sage: 5}},false,true)
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })
> db.student.find()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 30 }
{ "_id" : ObjectId("60127104939843297efc4bc6"), "sname" : "wangwu", "sage" : 25
}
{ "_id" : ObjectId("6012710f939843297efc4bc7"), "sname" : "chenliu", "sage" : 25
 }
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi", "sage" : 25
}
{ "_id" : ObjectId("60128e5b422ca12f6d93b597"), "sage" : "Tom" }
{ "_id" : ObjectId("601294f7422ca12f6d93b599"), "sname" : "aaa", "age" : 5 }
{ "_id" : ObjectId("60129689422ca12f6d93b59a"), "sage" : "lucy" }
```

4.5 集合文档基本操作

- 字段更新操作符

$rename

重命名字段，新的字段名称如果和文档中现有的字段名相同会覆盖

1、db.student.update({sname: 'aaa'}, {$rename : {sage: 'age'}})

将sname为‘aaa’的文档的sage键值名称改为age

```
> db.student.update({sname: 'aaa'}, {$rename : {sage: 'age'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.student.find()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 30 }
{ "_id" : ObjectId("60127104939843297efc4bc6"), "sname" : "wangwu", "sage" : 20
}
{ "_id" : ObjectId("6012710f939843297efc4bc7"), "sname" : "chenliu", "sage" : 20
 }
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi", "sage" : 25
}
{ "_id" : ObjectId("60128e5b422ca12f6d93b597"), "sname" : "Tom", "sage" : 25 }
{ "_id" : ObjectId("601294f7422ca12f6d93b599"), "sname" : "aaa", "age" : 5 }
```

2、db.student.update({sname: 'Tom'}, {$rename : {sname : 'sage'}})

由于Tom已经存在sage，把sname改为sage，就会覆盖掉原sage的数据造成数据丢失

```
> db.student.update({sname: 'Tom'}, {$rename : {sname : 'sage'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.student.find()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 30 }
{ "_id" : ObjectId("60127104939843297efc4bc6"), "sname" : "wangwu", "sage" : 20
}
{ "_id" : ObjectId("6012710f939843297efc4bc7"), "sname" : "chenliu", "sage" : 20
 }
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi", "sage" : 25
}
{ "_id" : ObjectId("60128e5b422ca12f6d93b597"), "sage" : "Tom" }
{ "_id" : ObjectId("601294f7422ca12f6d93b599"), "sname" : "aaa", "age" : 5 }
```

3、db.student.update({sname: 'lucy'}, {$rename : {sname : 'sage'}},true,false)

操作生效 upsert=true，先创建sname: 'lucy'，然后将sname改为 'sage'

```
> db.student.update({sname: 'lucy'}, {$rename : {sname : 'sage'}},true,false)
WriteResult({
        "nMatched" : 0,
        "nUpserted" : 1,
        "nModified" : 0,
        "_id" : ObjectId("60129689422ca12f6d93b59a")
})
> db.student.find()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 30 }
{ "_id" : ObjectId("60127104939843297efc4bc6"), "sname" : "wangwu", "sage" : 20
}
{ "_id" : ObjectId("6012710f939843297efc4bc7"), "sname" : "chenliu", "sage" : 20
 }
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi", "sage" : 25
}
{ "_id" : ObjectId("60128e5b422ca12f6d93b597"), "sage" : "Tom" }
{ "_id" : ObjectId("601294f7422ca12f6d93b599"), "sname" : "aaa", "age" : 5 }
{ "_id" : ObjectId("60129689422ca12f6d93b59a"), "sage" : "lucy" }
```

4.5 集合文档基本操作

- 删除数据

db.youCollection.remove()

三种用法

1、db.student.remove({})

删除student集合中的所有文档，保留索引

```
> db.student.remove({})
WriteResult({ "nRemoved" : 5 })
> show collections
student
system.indexes
teacher
> db.student.find()
>
```

```
> db.student.find()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 30 }
{ "_id" : ObjectId("60127104939843297efc4bc6"), "sname" : "wangwu", "sage" : 25
}
{ "_id" : ObjectId("6012710f939843297efc4bc7"), "sname" : "chenliu", "sage" : 25
 }
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi", "sage" : 25
}
{ "_id" : ObjectId("60128e5b422ca12f6d93b597"), "sage" : "Tom" }
{ "_id" : ObjectId("601294f7422ca12f6d93b599"), "sname" : "aaa", "age" : 5 }
{ "_id" : ObjectId("60129689422ca12f6d93b59a"), "sage" : "lucy" }
```

2、 db.student.remove({sage:25},1)

- 条件删除，仅删除一条sage=25的文档

```
> db.student.remove({sage:25},1)
WriteResult({ "nRemoved" : 1 })
> db.student.find()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 30 }
{ "_id" : ObjectId("6012710f939843297efc4bc7"), "sname" : "chenliu", "sage" : 25
 }
{ "_id" : ObjectId("601275ce939843297efc4bc8"), "sname" : "zhaoqi", "sage" : 25
}
{ "_id" : ObjectId("60128e5b422ca12f6d93b597"), "sage" : "Tom" }
{ "_id" : ObjectId("601294f7422ca12f6d93b599"), "sname" : "aaa", "age" : 5 }
{ "_id" : ObjectId("60129689422ca12f6d93b59a"), "sage" : "lucy" }
```

3、db.student.remove({sage:25})

- 条件删除，删除全部sage=25的文档

```
> db.student.remove({sage:25})
WriteResult({ "nRemoved" : 2 })
> db.student.find()
{ "_id" : 1, "sname" : "zhangsan", "sage" : 22 }
{ "_id" : ObjectId("601270fa939843297efc4bc5"), "sname" : "lisi", "sage" : 30 }
{ "_id" : ObjectId("60128e5b422ca12f6d93b597"), "sage" : "Tom" }
{ "_id" : ObjectId("601294f7422ca12f6d93b599"), "sname" : "aaa", "age" : 5 }
{ "_id" : ObjectId("60129689422ca12f6d93b59a"), "sage" : "lucy" }
```

4.5 集合文档基本操作

- 删除集合

db. youCollection.drop()

删除整个集合，数据、索引一起删除

1、db.student.remove({})

```
> db.student.remove({})
WriteResult({ "nRemoved" : 5 })
> show collections
student
system.indexes
teacher
> db.student.find()
>
```

2、 db. student. drop()

```
> db.student.drop()
true
> show collections
system.indexes
teacher
> db.student.find()
```

4.6 使用Java程序访问MongoDB

- 使用Java程序访问MongoDB

1. 环境配置

在Java程序中如果要使用MongoDB，需要确保已经安装了Java环境及MongoDB JDBC 驱动

第一步：下载mongo jar包，

下载地址（另存为）：

http://central.maven.org/maven2/org/mongodb/mongo-java-driver/3.2.2/mongo-java-driver-3.2.2.jar

需要将mongo.jar包含在你的 classpath 中

4.6 使用Java程序访问MongoDB

第二步：打开Eclipse，新建Java Project，新建Class，引入下载的jar包

第三步：编码实现

```
import java.util.ArrayList;
import java.util.List;
import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Filters;
```

2. 连接数据库

```
import com.mongodb.MongoClient;
……//这里省略其他需要导入的包
public class MongoDBJDBC{
  public static void main( String args[] ){
    try{
                        // 连接到 mongodb 服务
      MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
      // 连接到数据库
      DB db = mongoClient.getDB( "test" );
                        System.out.println("Connect to database successfully");
      boolean auth = db.authenticate(myUserName, myPassword);
                        System.out.println("Authentication: "+auth);
    }catch(Exception e){
              System.err.println( e.getClass().getName() + ": " + e.getMessage() );}
  }    }
```

3. 创建集合
可用com.mongodb.DB类中的createCollection()来创建集合

```
public class MongoDBJDBC{
  public static void main( String args[] ){
    try{
            // 连接到 mongodb 服务
      MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
      // 连接到数据库
      DB db = mongoClient.getDB( "test" );
            System.out.println("Connect to database successfully");
      boolean auth = db.authenticate(myUserName, myPassword);
            System.out.println("Authentication: "+auth);
      DBCollection coll = db.createCollection("mycol");
      System.out.println("Collection created successfully");
    }catch(Exception e){
            System.err.println( e.getClass().getName() + ": " + e.getMessage() );}
  }       }
```

4. 插入文档

可用com.mongodb.DBCollection类的 insert() 方法来插入文档

```
public class MongoDBJDBC{
  public static void main( String args[] ){
    try{
            // 连接到 mongodb 服务
      MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
      // 连接到数据库
      DB db = mongoClient.getDB( "test" );
            System.out.println("Connect to database successfully");
      boolean auth = db.authenticate(myUserName, myPassword);
            System.out.println("Authentication: "+auth);
      DBCollection coll = db.getCollection("mycol");
      System.out.println("Collection mycol selected successfully");
      BasicDBObject doc = new BasicDBObject("title", "MongoDB").
        append("description", "database").
        append("likes", 100).
        append("url", "http://www.w3cschool.cc/mongodb/").
        append("by", "w3cschool.cc");
        coll.insert(doc);
      System.out.println("Document inserted successfully");
    }catch(Exception e){
              System.err.println( e.getClass().getName() + ": " + e.getMessage() );}
  }    }
```

总结

- 本章内容

4.1 MongoDB简介

4.1.1 MongoDB的起源

4.1.2 MongoDB的特点

4.1.3 MongoDB适用场景

4.2 MongoDB的安装

4.3 MongoDB的概念介绍

4.3.1 文档

4.3.2 集合

4.3.3 数据库

总结

4.4 MongoDB常用命令介绍

4.5 集合文档基本操作

4.5.1 插入

4.5.2 查找

4.5.4 更新

4.5.5 删除

4.6 使用Java程序访问MongoDB