**MongoDB**

传智播客.黑马程序员

# 课程目标

目标1：熟练编写MongoDB增删改查的命令

目标2：java连接MongoDB并实现增删改查

# 1.MongoDB简介

## 1.1什么是MongoDB

MongoDB 是一个跨平台的，面向文档的数据库，是当前 NoSQL 数据库产品中最热门的一种。它介于关系数据库和非关系数据库之间，是非关系数据库当中功能最丰富，最像关系数据库的产品。它支持的数据结构非常松散，是类似JSON 的 BSON 格式，因此可以存储比较复杂的数据类型。

MongoDB 的官方网站地址是：<http://www.mongodb.org/>



## 1.2 MongoDB特点

MongoDB 最大的特点是他支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。它是一个面向集合的,模式自由的文档型数据库。

具体特点总结如下：

（1）面向集合存储，易于存储对象类型的数据

（2）模式自由

（3）支持动态查询

（4）支持完全索引，包含内部对象

（5）支持复制和故障恢复

（6）使用高效的二进制数据存储，包括大型对象（如视频等）

（7）自动处理碎片，以支持云计算层次的扩展性

（8）支持 Python，PHP，Ruby，Java，C，C#，Javascript，Perl 及 C++语言的驱动程序，社区中也提供了对 Erlang 及.NET 等平台的驱动程序

（9） 文件存储格式为 BSON（一种 JSON 的扩展）

## 1.3 MongoDB体系结构

MongoDB 的逻辑结构是一种层次结构。主要由：

文档(document)、集合(collection)、数据库(database)这三部分组成的。逻辑结构是面向用户

的，用户使用 MongoDB 开发应用程序使用的就是逻辑结构。

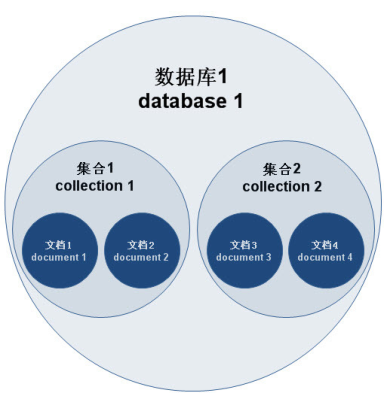
（1）MongoDB 的文档（document），相当于关系数据库中的一行记录。

（2）多个文档组成一个集合（collection），相当于关系数据库的表。

（3）多个集合（collection），逻辑上组织在一起，就是数据库（database）。

（4）一个 MongoDB 实例支持多个数据库（database）。

文档(document)、集合(collection)、数据库(database)的层次结构如下图:



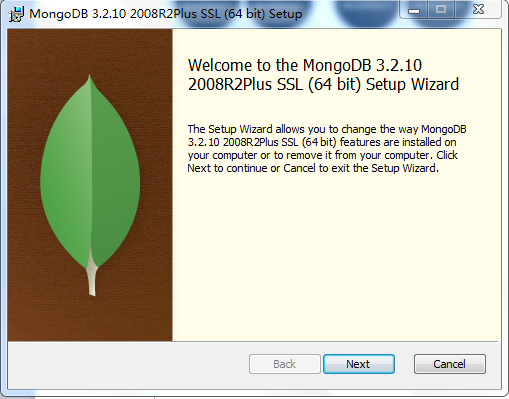
下表是MongoDB与MySQL数据库逻辑结构概念的对比

|  |  |
| --- | --- |
| MongoDb | 关系型数据库Mysql |
| 数据库(databases) | 数据库(databases) |
| 集合(collections) | 表(table) |
| 文档(document) | 行(row) |

# 2.安装与启动

## 2.1安装设置

双击“资源”中的“mongodb-win32-x86\_64-2008plus-ssl-3.2.10-signed.msi”

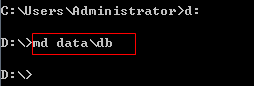


按照提示步骤安装即可。安装完成后，软件会安装在C:\Program Files\MongoDB 目录中。

我们要启动的服务程序就是C:\Program Files\MongoDB\Server\3.2\bin目录下的mongod.exe，为了方便我们每次启动，我将C:\Program Files\MongoDB\Server\3.2\bin 设置到**环境变量path**中。

## 2.2启动服务

（1）首先打开命令提示符，创建一个用于存放数据的目录

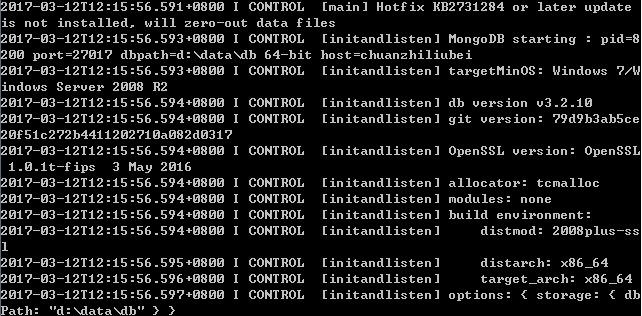


（2）启动服务



dbpath参数用于指定数据存储目录

启动后效果如下：



我们在启动信息中可以看到，mongoDB的默认端口是27017

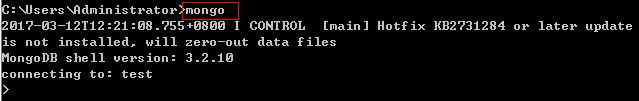


如果我们不想按照默认端口启动，可以通过--port 命令来修改端口



## 2.3登陆系统

我们另外打开命令提示符窗口，如果mongoDB是按默认的端口启动的，并且是部署在本机的。输入命令 mongo 即可登陆系统



从界面输出的信息我们可以得知，它默认连接的是test数据库

如果是要连接远程的mongoDB服务器 ，就输入命令

|  |
| --- |
| mongo 远程IP地址 |

如果远程的mongoDB服务端口不是默认的，需要输入命令

|  |
| --- |
| mongo 远程IP地址：端口 |

输入exit命令可退回到命令提示符

# 3.基本增删改查操作

## 3.1选择或创建数据库

使用use 数据库名称即可选择数据库，如果该数据库不存在会自动创建



## 3.2插入文档

文档相当于关系数据库中的记录

首先我们定义一个文档变量，格式为变量名称={}; 例如：



接下来就是将这个变量存入MongoDB

格式为：

|  |
| --- |
| db.集合名称.save(变量); |

这里的集合就相当于关系数据库中的表。例如：



这样就在student集合中存入文档。如果这个student集合不存在，就会自动创建。

当然，你也可以不用定义变量，直接把变量值放入save方法中也是可以地。



为了方便后期测试，我们再多加点数据

|  |
| --- |
| db.student.save({name:"沙和尚",sex:"男",age:25,address:"流沙河路11号"});  db.student.save({name:"唐僧",sex:"男",age:35,address:"东土大唐"});  db.student.save({name:"白骨精",sex:"女",age:18,address:"白骨洞"});  db.student.save({name:"白龙马",sex:"男",age:20,address:"西海"});  db.student.save({name:"哪吒",sex:"男",age:15,address:"莲花湾小区"}); |

## 3.3查询集合

我们要查询某集合的所有文档，使用find()方法。语法格式为：

|  |
| --- |
| db.集合名称.find(); |

例如，我们要查询student集合中的所有文档：



这里你会发现每条文档会有一个叫\_id的字段，这个相当于我们原来关系数据库中表的主键，当你在插入文档记录时没有指定该字段，MongDB会自动创建，其类型是ObjectID类型。

如果我们在插入文档记录时指定该字段也可以，其类型可以使ObjectID类型，也可以是MongoDB支持的任意类型. 例如：

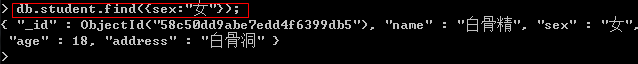


我们再次查询

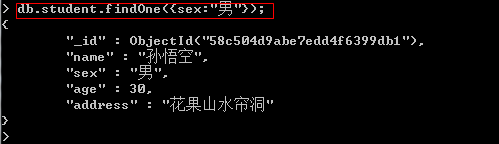


如果我想按一定条件来查询，比如我想查询性别为“女”的记录，怎么办？很简单！

只要在find()中添加参数即可，参数也是json格式，如下：

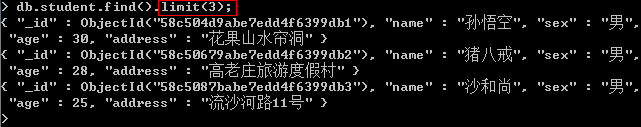


为了避免游标可能带来的开销，MongoDB还提供了一个叫findOne()的方法，用来返回结果集的第一条记录。



性别为男的有很多条，这里只返回了第一条记录。

当我们需要返回查询结果的前几条记录时，可以使用limit方法，例如：



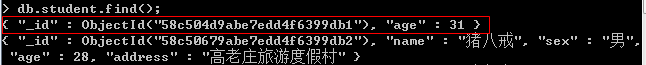
## 3.4修改文档

我们要想修改记录，可以使用update方法 .

例如：我向将姓名为孙悟空的学员文档中的age字段值改为31，执行下列语句，看会发生什么？



再次查询：



哦，悲剧了~~ 原来的孙悟空的文档只剩下\_id 和age两个字段了。

那如何保留其它字段值呢？

我们需要使用MongoDB提供的修改器$set 来实现，请看下列代码。



再次查询，会发现“猪八戒”文档中原有的其它字段还保留下来，而更新age字段也成功了。

## 3.5删除文档

删除文档使用remove()方法，格式为：

|  |
| --- |
| db.集合名称.remove( 条件 ); |

请慎用remove({}), 它会一条不剩地把你的集合所有文档删的干干净净。

我们现在演示一下，删除name为“哪吒”的记录：



再次查询，会发现哪吒的文档不见了。

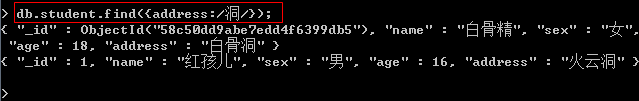
# 4.高级查询

## 4.1模糊查询

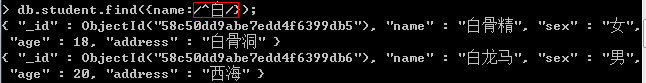
MongoDB的模糊查询是通过正则表达式的方式实现的。格式为：

/模糊查询字符串/

例如，我要查询student集合中address字段中含有“洞”的所有文档，代码如下：



如果要查询name字段中以“白”开头的，代码如下：



## 4.2 Null值处理

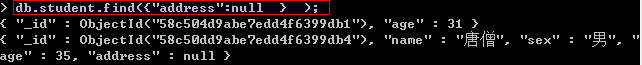
如果我们想找出集合中某字段值为空的文档，如何查询呢？其实和我们之前的条件查询是一样的，条件值写为null就可以了。

我们现在集合中的文档都是没有空值的，为了方便测试，现在我们将数据做些修改：

将“唐僧”的address改为空



再次查询：



我们会发现不仅会显示“唐僧”这条文档，之前因为修改导致address字段丢失的那条记录也出现了。也就是说，这种查询会查询出该字段为null的以及不存在该字段的文档记录。

## 4.3大于小于

<, <=, >, >= 这个操作符也是很常用的，格式如下

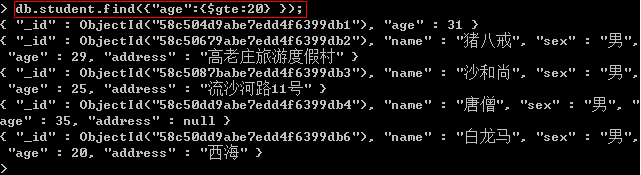
db.collection.find({ "field" : { $gt: value } } ); // 大于: field > value

db.collection.find({ "field" : { $lt: value } } ); // 小于: field < value

db.collection.find({ "field" : { $gte: value } } ); // 大于等于: field >= value

db.collection.find({ "field" : { $lte: value } } ); // 小于等于: field <= value

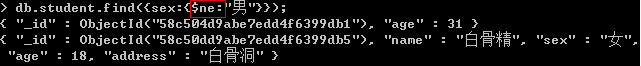
示例：查询年龄大于等于20岁的学员记录



## 4.4不等于

不等于使用$ne操作符。

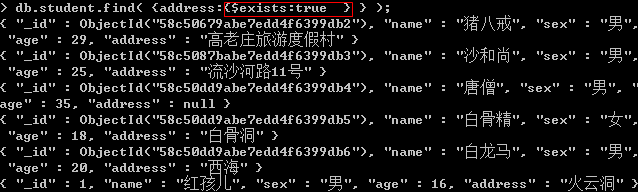
示例：查询sex字段不为“男”的文档



## 4.5判断字段是否存在

判断字段是否存在使用$exists操作符。

示例：查询所有含有address字符的文档。



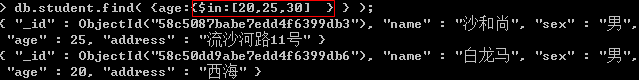
示例：查询所有不含有address字符的文档。



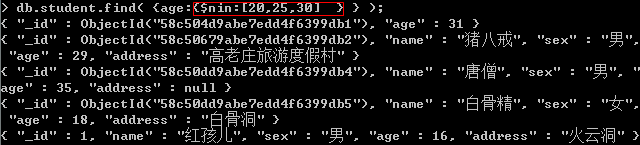
## 4.6包含与不包含

包含使用$in操作符。

示例：查询student集合中age字段包含20,25,30的文档



示例：查询student集合中age字段不包含20,25,30的文档



## 4.7统计记录条数

统计记录条件使用count()方法。

示例：查询student集合的文档条数。



示例：查询student集合中age字段小于等于20的文档条数。

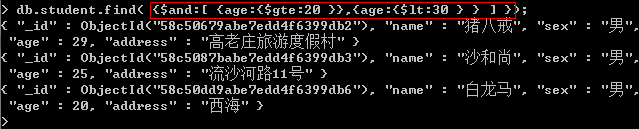


## 4.8 条件连接--并且

我们如果需要查询同时满足两个以上条件，需要使用$and操作符将条件进行关联。（相当于SQL的and）

格式为：$and:[ { },{ },{ } ]

示例：查询student集合中age大于等于20 并且age小于30的文档

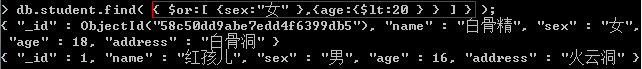


## 4.9 条件连接--或者

如果两个以上条件之间是或者的关系，我们使用$or操作符进行关联，与前面$and的使用方式相同

格式为：$or:[ { },{ },{ } ]

示例：查询student集合中sex为女，或者年龄小于20的文档记录



# 5.java连接MongoDB

## 5.1查询文档

### 5.1.1查询全部记录

（1）创建maven工程mongoDBDemo ，引入依赖。

|  |
| --- |
| <dependencies>  <dependency>  <groupId>org.mongodb</groupId>  <artifactId>mongodb-driver</artifactId>  <version>3.4.1</version>  </dependency>  </dependencies> |

（2）编写代码，遍历student集合所有数据：

|  |
| --- |
| MongoClient client=**new** MongoClient();//创建连接对象  MongoDatabase database = client.getDatabase("itcastdb");//获取数据库  MongoCollection<Document> collection = database.getCollection("student");//获取集合    FindIterable<Document> list = collection.find();//获取文档集合  **for**( Document doc: list){//遍历集合中的文档输出数据  System.***out***.println("name:"+ doc.getString("name") );  System.***out***.println("sex:"+ doc.getString("sex") );  System.***out***.println("age:"+ doc.getDouble("age") );//默认为浮点型  System.***out***.println("address:"+ doc.getString("address") );  System.***out***.println("--------------------------");  } |

MongoDB的数字类型默认使用64位浮点型数值。{“x”：3.14}或{“x”：3}。对于整型值，可以使用NumberInt（4字节符号整数），{“x”:NumberInt(“3”)} 或NumberLong（8字节符号整数）{“x”:NumberLong(“3”)}

### 5.1.2匹配查询

MongoDB使用BasicDBObject类型封装查询条件，构造方法的参数为key 和value .

示例：查询student集合中name为猪八戒的文档

|  |
| --- |
| //构建查询条件  BasicDBObject **bson**=**new** BasicDBObject("name", "猪八戒");  FindIterable<Document> list = collection.find(**bson**);//获取文档集合  //....遍历集合 |

### 5.1.3模糊查询

构建模糊查询条件是通过正则表达式的方式来实现的

（1）完全匹配Pattern pattern = Pattern.compile("^name$");

（2）右匹配Pattern pattern = Pattern.compile("^.\*name$");

（3）左匹配Pattern pattern = Pattern.compile("^name.\*$");

（4）模糊匹配Pattern pattern = Pattern.compile("^.\*name.\*$");

示例：模糊查询student集合中address 中含有洞的文档记录

|  |
| --- |
| //模糊查询：like %洞%  Pattern queryPattern = Pattern.*compile*("^.\*洞.\*$");  BasicDBObject bson=**new** BasicDBObject("address", queryPattern);  FindIterable<Document> list = collection.find(bson);//获取文档集合  //....遍历集合 |

### 5.1.4大于小于

在MongoDB提示符下条件json字符串为{ age: { $lt :20 } } ，对应的java代码也是BasicDBObject 的嵌套。

示例：查询student集合中age小于20的文档记录

|  |
| --- |
| //查询年龄小于20的  BasicDBObject bson=**new** BasicDBObject("age", **new** BasicDBObject("$lt",20));  FindIterable<Document> list = collection.find(bson);//获取文档集  //....遍历集合 |

### 5.1.5条件连接--并且

示例：查询年龄大于等于20并且小于30的文档记录

|  |
| --- |
| //查询年龄大于等于20的  BasicDBObject bson1=**new** BasicDBObject("age", **new** BasicDBObject("$gte",20));  //查询年龄小于30的  BasicDBObject bson2=**new** BasicDBObject("age", **new** BasicDBObject("$lt",30));  //构建查询条件and  BasicDBObject bson=**new** BasicDBObject("$and", Arrays.*asList*(bson1,bson2) ); |

### 5.1.6条件连接--或者

示例：查询年龄小于等于20或者性别为女的文档记录

|  |
| --- |
| BasicDBObject bson1=**new** BasicDBObject("age", **new** BasicDBObject("$lte",20));  BasicDBObject bson2=**new** BasicDBObject("sex", "女");  //构建查询条件or  BasicDBObject bson=**new** BasicDBObject("$or", Arrays.*asList*( bson1, bson2 ) ); |

## 5.2增加文档

我们使用insertOne方法来插入文档。

示例：添加文档记录--名称：铁扇公主 性别:女 年龄：28 地址：芭蕉洞

|  |
| --- |
| //获取连接  MongoClient client=**new** MongoClient();  //得到数据库  MongoDatabase database = client.getDatabase("itcastdb");  //得到集合封装对象  MongoCollection<Document> collection = database.getCollection("student");  Map<String, Object> map=**new** HashMap();  map.put("name", "铁扇公主");  map.put("sex", "女");  map.put("age", 35.0);  map.put("address", "芭蕉洞");  Document doc=**new** Document(map);  collection.insertOne(doc);//插入一条记录  //collection.insertMany(documents);//一次性插入多条文档 |

## 5.3删除文档

示例：将名称为铁扇公主的文档删除

|  |
| --- |
| //获取连接  MongoClient client=**new** MongoClient();  //得到数据库  MongoDatabase database = client.getDatabase("itcastdb");  //得到集合封装对象  MongoCollection<Document> collection = database.getCollection("student");  BasicDBObject bson=**new** BasicDBObject("name", "铁扇公主");  collection.deleteOne(bson);//删除记录（符合条件的第一条记录）  //collection.deleteMany(bson);//删除符合条件的全部记录 |

## 5.4修改文档

示例：将红孩儿的地址修改为“南海”

|  |
| --- |
| //获取连接  MongoClient client=**new** MongoClient();  //得到数据库  MongoDatabase database = client.getDatabase("itcastdb");  //得到集合封装对象  MongoCollection<Document> collection = database.getCollection("student");  //修改的条件  BasicDBObject bson= **new** BasicDBObject("name", "红孩儿");  //修改后的值  BasicDBObject bson2 = **new** BasicDBObject("$set",**new** BasicDBObject("address", "南海"));  //参数1：修改条件 参数2：修改后的值  collection.updateOne(bson, bson2);  //collection.updateMany(filter, update);//修改符合条件的所有记录 |

updateMany方法用于修改符合条件的所有记录

updateOne方法用于修改符合条件的第一条记录

# 6.MongoDB连接池

## 6.1代码实现

MongoClient 被设计为线程安全的类，也就是我们在使用该类时不需要考虑并发的情况，这样我们可以考虑把MongoClient 做成一个静态变量，为所有线程公用，不必每次都销毁。这样可以极大提高执行效率。实际上，这是MongoDB提供的内置的连接池来实现的。

首先我们先创建一个“管理类”，相当于我们原来的BaseDao

|  |
| --- |
| **package** cn.itcast.demo;  **import** com.mongodb.MongoClient;  **import** com.mongodb.MongoClientOptions;  **import** com.mongodb.MongoClientOptions.Builder;  **import** com.mongodb.WriteConcern;  **import** com.mongodb.client.MongoDatabase;  **public** **class** MongoManager {  **private** **static** MongoClient *mongoClient*=**null**;    //对mongoClient初始化  **private** **static** **void** init(){  *mongoClient*=**new** MongoClient();  }    **public** **static** MongoDatabase getDatabase(){  **if**(*mongoClient*==**null**){  *init*();  }  **return** *mongoClient*.getDatabase("itcastdb");  }  } |

然后我们创建一个StudentDao

|  |
| --- |
| **package** cn.itcast.demo;  **import** org.bson.Document;  **import** com.mongodb.client.MongoCollection;  **import** com.mongodb.client.MongoDatabase;  /\*\*  \* 学员数据访问层  \* **@author** Administrator  \*  \*/  **public** **class** StudentDao {  **public** **void** save(String name,String sex,**double** age,String address){  MongoDatabase database = MongoManager.*getDatabase*();  MongoCollection<Document> collection = database.getCollection("student2");  Document docment=**new** Document();  docment.put("name", name);  docment.put("sex", sex);  docment.put("age", age);  docment.put("address", address);  collection.insertOne(docment);  }  } |

我们现在做个测试，循环插入2万条数据，看看执行时间是多长时间

|  |
| --- |
| **package** cn.itcast.demo;  **import** java.util.Date;  **public** **class** TestPool {  **public** **static** **void** main(String[] args) {  **long** startTime = **new** Date().getTime();//开始时间    StudentDao studentDao=**new** StudentDao();  **for**(**int** i=0;i<20000;i++){  studentDao.save("测试"+i, "男", 25.0, "测试地址"+i);  }  **long** endTime = **new** Date().getTime();//完成时间  System.***out***.println("完成时间："+(endTime-startTime)+"毫秒");  }  } |

经过测试：所用毫秒数为3589

## 6.2参数设置

我们在刚才的代码基础上进行连接池参数的设置

修改MongoManager的init方法

|  |
| --- |
| //对mongoClient初始化  **private** **static** **void** init(){  //连接池选项  Builder builder = **new** MongoClientOptions.Builder();//选项构建者  builder.connectTimeout(5000);//设置连接超时时间  builder.socketTimeout(5000);//读取数据的超时时间  builder.connectionsPerHost(30);//每个地址最大请求数  builder.writeConcern(WriteConcern.~~NORMAL~~);//写入策略，仅抛出网络异常  MongoClientOptions options = builder.build();  *mongoClient*=**new** MongoClient("127.0.0.1",options);  } |

再次进行测试：所用的毫秒1544

下面是写入策略。

WriteConcern.NONE:没有异常抛出

WriteConcern.NORMAL:仅抛出网络错误异常，没有服务器错误异常

WriteConcern.SAFE:抛出网络错误异常、服务器错误异常；并等待服务器完成写操作。

WriteConcern.MAJORITY: 抛出网络错误异常、服务器错误异常；并等待一个主服务器完成写操作。

WriteConcern.FSYNC\_SAFE: 抛出网络错误异常、服务器错误异常；写操作等待服务器将数据刷新到磁盘。

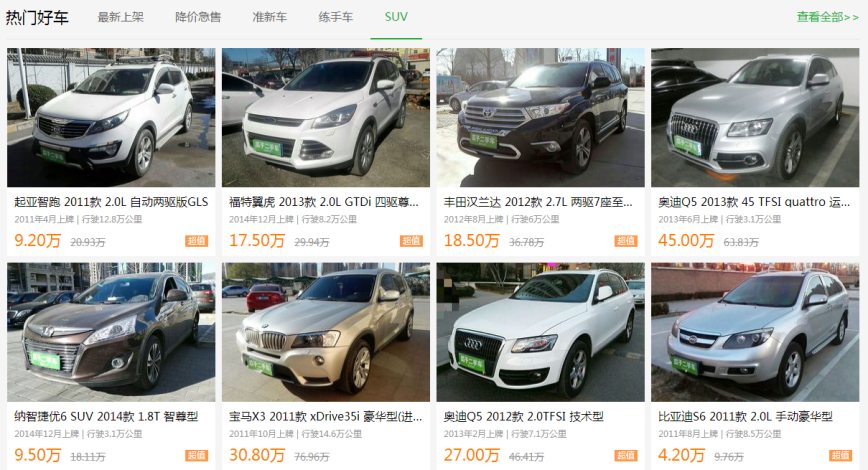
WriteConcern.JOURNAL\_SAFE:抛出网络错误异常、服务器错误异常；写操作等待服务器提交到磁盘的日志文件。

WriteConcern.REPLICAS\_SAFE:抛出网络错误异常、服务器错误异常；等待至少2台服务器完成写操作。

# 7.综合案例-《网站点击日志分析组件》

## 7.1需求分析

《花生二手车》交易网站日访问IP高达2万+ ，每秒点击频率在2000次左右。为了能够对访问用户的行为做进一步的分析，产品部提出需求，用户每次点击浏览二手车都要记录该用户ID、访问IP、访问时间、点击车型、点击商品ID、价格等信息。



## 7.2数据库设计

浏览日志browseLog

|  |  |  |
| --- | --- | --- |
| 字段名称 | 字段类型 | 字段含义 |
| userid | 字符 | 用户ID |
| ip | 字符 | 访问IP |
| browseTime | 时间 | 访问时间 |
| model | 字符 | 点击车型 |
| goodsid | 字符 | 点击商品ID |
| price | 数值 | 价格 |
| remark | 字符 | 备注 |

## 7.3日志写入

（1）创建工程sitelog ，在pom.xml中引入依赖。

|  |
| --- |
| <dependencies>  <dependency>  <groupId>org.mongodb</groupId>  <artifactId>mongodb-driver</artifactId>  <version>3.4.1</version>  </dependency>  </dependencies> |

（2）在src/main/resources 添加配置文件sitelog.properties

|  |
| --- |
| host=127.0.0.1  port=27017 |

这个配置文件用于配置主机地址和端口

1. 创建包com.huasheng.sitelog，建立Config 类，用于读取配置文件

|  |
| --- |
| **package** com.huasheng.sitelog;  **import** java.io.IOException;  **import** java.io.InputStream;  **import** java.util.Properties;  /\*\*  \* 配置类  \* **@author** Administrator  \*  \*/  **public** **class** Config {  **static**{  **try** {  Properties p=**new** Properties();  InputStream input=Config.**class**.getResourceAsStream("/sitelog.propertis");  p.load(input);  *host*=p.getProperty("host");  *port*=Integer.*parseInt*( p.getProperty("port"));  input.close();  } **catch** (IOException e) {  e.printStackTrace();  }//加载  }  **private** **static** String *host*;//主机地址  **private** **static** **int** *port*;//端口    **public** **static** String getHost() {  **return** *host*;  }  **public** **static** **int** getPort() {  **return** *port*;  }  } |

（4）创建管理类

|  |
| --- |
| **package** com.huasheng.sitelog;  **import** com.mongodb.MongoClient;  **import** com.mongodb.MongoClientOptions;  **import** com.mongodb.MongoClientOptions.Builder;  **import** com.mongodb.ServerAddress;  **import** com.mongodb.WriteConcern;  **import** com.mongodb.client.MongoDatabase;  /\*\*  \* Mongo数据库连接管理类  \* **@author** Administrator  \*  \*/  **public** **class** MongoManager {    **private** **static** MongoClient *mongoClient*=**null**;    //初始化  **private** **static** **void** init(){  //创建一个选项构造器  Builder builder = **new** MongoClientOptions.Builder();  builder.connectTimeout(5000);//设置连接超时时间  builder.socketTimeout(5000);//读取数据的超时时间  builder.connectionsPerHost(30);//设置每个地址最大连接数  builder.writeConcern(WriteConcern.~~NORMAL~~);//设置写入策略 ,只有网络异常才会抛出  //得到选项封装  MongoClientOptions options = builder.build();  *mongoClient*=**new** MongoClient(**new** ServerAddress(Config.*getHost*(), Config.*getPort*()),options);  }    **public** **static** MongoDatabase getDatabase(){  **if**(*mongoClient*==**null**){  *init*();  }  **return** *mongoClient*.getDatabase("itcastdb");  }  } |

1. 日志工具类

|  |
| --- |
| **package** com.huasheng.sitelog;  **import** java.util.Map;  **import** org.bson.Document;  **import** com.mongodb.client.MongoCollection;  **import** com.mongodb.client.MongoDatabase;  /\*\*  \* 站点日志数据工具类  \* **@author** Administrator  \*  \*/  **public** **class** SiteLogUtil {    /\*\*  \* 写入日志  \* **@param** logname 日志名称  \* **@param** map 日志数据  \*/  **public** **static** **void** save(String logname,Map<String, Object> map){    MongoDatabase database = MongoManager.*getDatabase*();  MongoCollection<Document> collection = database.getCollection(logname);  Document doc=**new** Document(map);  collection.insertOne(doc);  }  } |

（6）编写测试代码

|  |
| --- |
| Map map=**new** HashMap();  map.put("userid", "8888");  map.put("ip", "188.188.200.2");  map.put("browseTime", **new** Date());  map.put("model", "大众");  map.put("goodsid", "123123");  map.put("price", 15.3);  map.put("remark", "八成新，贱卖了");  SiteLogUtil.*save*("browseLog", map);//存入日志 |

## 7.4日志查询

### 7.4.1条件查询

（1）在SiteLogUtil类中添加方法

|  |
| --- |
| /\*\*  \* 按条件查询  \* **@param** logName  \* **@param** map  \* **@return**  \*/  **public** **static** FindIterable<Document> list(String logName,Map<String, Object> map){  MongoDatabase database = MongoManager.*getDatabase*();  MongoCollection<Document> collection = database.getCollection(logName);  BasicDBObject bson=**new** BasicDBObject(map);//构建查询条件  **return** collection.find(bson);  } |

（2）编写测试代码

|  |
| --- |
| Map<String, Object> map =**new** HashMap();  map.put("userid", "8888");  FindIterable<Document> list = SiteLogUtil.*list*("browseLog", map);  String json = JSON.*serialize*(list);  System.***out***.println(json); |

### 7.4.2分页查询

（1）在SiteLogUtil类中添加方法

|  |
| --- |
| /\*\*  \* 分页查询日志  \* **@param** logName 日志名称  \* **@param** map 条件  \* **@param** pageIndex 页码  \* **@param** pageSize 页大小  \* **@return**  \*/  **public** **static** Map<String,Object> listPage(String logName,Map<String, Object> map,**int** pageIndex,**int** pageSize){  MongoDatabase database = MongoManager.*getDatabase*();  MongoCollection<Document> collection = database.getCollection(logName);  BasicDBObject bson=**new** BasicDBObject(map);//构建查询条件  FindIterable<Document> find = collection.find(bson);  **int** skip= (pageIndex-1)\*pageSize;  find.skip( skip);//跳过记录数  find.limit(pageSize);//一页查询记录数  //{ total:x,rows:[] }  **long** count = collection.count(bson);  Map<String,Object> m=**new** HashMap();  m.put("total", count);  m.put("rows", find);  **return** m;  } |

（2）添加测试数据

|  |
| --- |
| **for**(**int** i=0;i<1000;i++){    Map<String, Object> map=**new** HashMap();  map.put("userid", "900"+i);//用户ID  map.put("ip", "121.211.112.212");  map.put("browseTime", **new** Date());//浏览时间  map.put("model", "大众"+i);//型号  map.put("goodsid", "123456");//商品ID  map.put("price", 11.8);//价格  map.put("remark", "八成新，快来买吧");    SiteLogUtil.*save*("browseLog", map);  } |

1. 编写测试代码：

|  |
| --- |
| Map<String, Object> map=**new** HashMap();  map.put("goodsid", "123456");  Map<String, Object> m = SiteLogUtil.*listPage*("browseLog", map, 2, 10);  String json = JSON.*serialize*(m);  System.***out***.println(json); |

使用Maven 的package命令进行打包。

创建WEB工程，引入jar包，调用此方法即可实现日志查询。代码略。