

## 第3章-文档型数据库MongoDB

学习目标：

- 理解MongoDb的特点和体系结构
- 掌握常用的MongoDB命令
- 能够运用Java操作MongoDB
- 使用SpringDataMongoDB完成吐槽微服务的开发

### 1 MongoDB简介

#### 1.1 吐槽和评论数据特点分析

吐槽和评论两项功能存在以下特点：

- (1) 数据量大
- (2) 写入操作频繁
- (3) 价值较低

对于这样的数据，我们更适合使用MongoDB来实现数据的存储

#### 1.2 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/>



## 1.3 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）

## 1.4 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)

## 1.5 数据类型

基本数据类型

null: 用于表示空值或者不存在的字段, `{"x":null}`

布尔型: 布尔类型有两个值true和false, `{"x":true}`

数值: shell默认使用64为浮点型数值。`{"x": 3.14}`或`{"x": 3}`。对于整型值, 可以使用NumberInt(4字节符号整数)或NumberLong(8字节符号整数), `{"x":NumberInt("3")}``{"x":NumberLong("3")}`

字符串: UTF-8字符串都可以表示为字符串类型的数据, `{"x": "呵呵"}`

日期: 日期被存储为自新纪元依赖经过的毫秒数, 不存储时区, `{"x":new Date()}`

正则表达式：查询时，使用正则表达式作为限定条件，语法与JavaScript的正则表达式相同，{"x":/[abc]/}

数组：数据列表或数据集可以表示为数组，{"x": ["a", "b", "c"]}

内嵌文档：文档可以嵌套其他文档，被嵌套的文档作为值来处理，{"x":{"y":3}}

对象id：对象id是一个12字节的字符串，是文档的唯一标识，{"x":objectId() }

二进制数据：二进制数据是一个任意字节的字符串。它不能直接在shell中使用。如果要将非utf-字符保存到数据库中，二进制数据是唯一的方式。

代码：查询和文档中可以包括任何JavaScript代码，{"x":function(){/.../}}

## 2 走进MongoDB

### 2.1 MongoDB安装与启动

#### 2.1.1 window系统MongoDB安装

安装

双击“资源\微服务相关\配套软件\mongodb”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。

启动

(1) 首先打开命令提示符，创建一个用于存放数据的目录

```
md d:\data
```

(2) 启动服务

```
mongod --dbpath=d:\data
```

我们在启动信息中可以看到，mongoDB的默认端口是27017

如果我们想改变默认的启动端口，可以通过--port来指定端口

在命令提示符输入以下命令即可完成登陆

```
mongo
```

退出mongodb

```
exit
```

## 1.5.2 Docker 环境下MongoDB安装

在宿主机创建mongo容器

```
docker run -di --name=tensquare_mongo -p 27017:27017 mongo
```

远程登陆

```
mongo 192.168.184.134
```

以吐槽表为例讲解MongoDB常用命令

吐槽表	spit		
字段名称	字段含义	字段类型	备注
<code>_id</code>	ID	文本	
<code>content</code>	吐槽内容	文本	
<code>publishtime</code>	发布日期	日期	
<code>userid</code>	发布人ID	文本	
<code>nickname</code>	发布人昵称	文本	
<code>visits</code>	浏览量	整型	
<code>thumbup</code>	点赞数	整型	
<code>share</code>	分享数	整型	
<code>comment</code>	回复数	整型	
<code>state</code>	是否可见	文本	
<code>parentid</code>	上级ID	文本	

## 2.2 常用命令

### 2.2.1 选择和创建数据库

选择和创建数据库的语法格式：

```
use 数据库名称
```

如果数据库不存在则自动创建

以下语句创建spit数据库

```
use spitdb
```

### 2.2.2 插入与查询文档

插入文档的语法格式：

```
db.集合名称.insert(数据);
```

我们这里可以插入以下测试数据：

```
db.spit.insert({content:"听说十次方课程很给力呀",userid:"1011",nickname:"小雅",visits:NumberInt(902)})
```

查询集合的语法格式：

```
db.集合名称.find()
```

如果我们要查询spit集合的所有文档，我们输入以下命令

```
db.spit.find()
```

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，MongoDB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，其类型可以是ObjectID类型，也可以是MongoDB支持的任意类型。

输入以下测试语句：

```
db.spit.insert({_id:"1",content:"我还是没有想明白到底为啥出错",userid:"1012",nickname:"小明",visits:NumberInt(2020)});
db.spit.insert({_id:"2",content:"加班到半夜",userid:"1013",nickname:"凯撒",visits:NumberInt(1023)});
db.spit.insert({_id:"3",content:"手机流量超了咋办?",userid:"1013",nickname:"凯撒",visits:NumberInt(111)});
db.spit.insert({_id:"4",content:"坚持就是胜利",userid:"1014",nickname:"诺诺",visits:NumberInt(1223)});
```

如果我想按一定条件来查询，比如我想查询userid为1013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：

```
db.spit.find({userid:'1013'})
```

如果你只需要返回符合条件的第一条数据，我们可以使用findOne命令来实现

```
db.spit.findOne({userid:'1013'})
```

如果你想返回指定条数的记录，可以在find方法后调用limit来返回结果，例如：

```
db.spit.find().limit(3)
```

## 2.2.3 修改与删除文档

修改文档的语法结构：

```
db.集合名称.update(条件,修改后的数据)
```

如果我们想修改\_id为1的记录，浏览量为1000，输入以下语句：

```
db.spit.update({_id:"1"},{visits:NumberInt(1000)})
```

执行后，我们会发现，这条文档除了visits字段其它字段都不见了，为了解决这个问题，我们需要使用修改器\$set来实现，命令如下：

```
db.spit.update({_id:"2"},{$set:{visits:NumberInt(2000)}})
```

这样就OK啦。

删除文档的语法结构：

```
db.集合名称.remove(条件)
```

以下语句可以将数据全部删除，请慎用

```
db.spit.remove({})
```

如果删除visits=1000的记录，输入以下语句

```
db.spit.remove({visits:1000})
```

## 2.2.4 统计条数

统计记录条件使用count()方法。以下语句统计spit集合的记录数



```
db.spit.count()
```

如果按条件统计，例如：统计userid为1013的记录条数

```
db.spit.count({userid:"1013"})
```

## 2.2.5 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，我要查询吐槽内容包含“流量”的所有文档，代码如下：

```
db.spit.find({content:/流量/})
```

如果要查询吐槽内容中以“加班”开头的，代码如下：

```
db.spit.find({content:/^加班/})
```

## 2.2.6 大于 小于 不等于

<, <=, >, >= 这个操作符也是很常用的，格式如下：

```
db.集合名称.find({ "field" : { $gt: value } }) // 大于: field > value  
db.集合名称.find({ "field" : { $lt: value } }) // 小于: field < value  
db.集合名称.find({ "field" : { $gte: value } }) // 大于等于: field >= value  
db.集合名称.find({ "field" : { $lte: value } }) // 小于等于: field <= value  
db.集合名称.find({ "field" : { $ne: value } }) // 不等于: field != value
```

示例：查询吐槽浏览量大于1000的记录

```
db.spit.find({visits:{$gt:1000}})
```

## 2.2.7 包含与不包含

包含使用\$in操作符。

示例：查询吐槽集合中userid字段包含1013和1014的文档

```
db.spit.find({userid:{$in:["1013","1014"]}})
```

不包含使用\$nin操作符。

示例：查询吐槽集合中userid字段不包含1013和1014的文档

```
db.spit.find({userid:{$nin:["1013","1014"]}})
```

## 2.2.8 条件连接

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：查询吐槽集合中visits大于等于1000 并且小于2000的文档

```
db.spit.find({$and:[ {visits:{$gte:1000}} ,{visits:{$lt:2000} } ]})
```

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：查询吐槽集合中userid为1013，或者浏览量小于2000的文档记录

```
db.spit.find({$or:[ {userid:"1013"} ,{visits:{$lt:2000} } ]})
```

## 2.2.9 列值增长

如果我们想实现对某列值在原有值的基础上进行增加或减少，可以使用\$inc运算符来实现

```
db.spit.update({_id:"2"},{$inc:{visits:NumberInt(1)}} )
```

## 3 Java操作MongoDB

### 3.1 mongodb-driver

mongodb-driver是mongo官方推出的java连接mongoDB的驱动包，相当于JDBC驱动。我们通过一个入门的案例来了解mongodb-driver的基本使用

#### 3.1.1 查询全部记录

(1) 创建工程 mongoDemo, 引入依赖

```
<dependencies>
  <dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver</artifactId>
    <version>3.6.3</version>
  </dependency>
</dependencies>
```

(2) 创建测试类



```
/**
 * MongoDB入门小demo
 */
public class MongoDemo {

    public static void main(String[] args) {
        MongoClient client=new MongoClient("192.168.184.134");//创建连接
        MongoDBDatabase spitdb = client.getDatabase("spitdb");//打开数据库
        MongoCollection<Document> spit = spitdb.getCollection("spit");//
        获取集合
        FindIterable<Document> documents = spit.find();//查询记录获取文档集
        合
        for(Document document:documents){ //
            System.out.println("内容: "+ document.getString("content"));
            System.out.println("用户ID:"+document.getString("userid"));
            System.out.println("浏览量: "+document.getInteger("visits"));
        }
        client.close();//关闭连接
    }
}
```

### 3.1.2 条件查询

BasicDBObject对象：表示一个具体的记录，BasicDBObject实现了DBObject，是key-value的数据结构，用起来和HashMap是基本一致的。

(1) 查询userid为1013的记录



```
public class MongoDemo1 {  
  
    public static void main(String[] args) {  
  
        MongoClient client=new MongoClient("192.168.184.134");//创建连接  
        MongoDBDatabase spitdb = client.getDatabase("spitdb");//打开数据库  
        MongoCollection<Document> spit = spitdb.getCollection("spit");//  
        获取集合  
        BasicDBObject bson=new BasicDBObject("userid","1013");// 构建查询  
        条件  
        FindIterable<Document> documents = spit.find(bson);//查询记录获取结  
        果集合  
        for(Document document:documents){ //  
            System.out.println("内容: "+ document.getString("content"));  
            System.out.println("用户ID:"+document.getString("userid"));  
            System.out.println("浏览量: "+document.getInteger("visits"));  
        }  
        client.close();//关闭连接  
    }  
}
```

(2) 查询浏览量大于1000的记录



```
public class MongoDemo2 {  
  
    public static void main(String[] args) {  
  
        MongoClient client=new MongoClient("192.168.184.134");//创建连接  
        MongoDBDatabase spitdb = client.getDatabase("spitdb");//打开数据库  
        MongoCollection<Document> spit = spitdb.getCollection("spit");//  
        获取集合  
  
        BasicDBObject bson=new BasicDBObject("visits",new  
        BasicDBObject("$gt",1000) );// 构建查询条件  
  
        FindIterable<Document> documents = spit.find(bson);//查询记录获取结  
        果集合  
        for(Document document:documents){ //  
            System.out.println("内容: "+ document.getString("content"));  
            System.out.println("用户ID:"+document.getString("userid"));  
            System.out.println("浏览量: "+document.getInteger("visits"));  
        }  
        client.close();//关闭连接  
    }  
}
```

### 3.1.3 插入数据

```
public class MongoDemo3 {  
  
    public static void main(String[] args) {  
        MongoClient client=new MongoClient("192.168.184.134");//创建连接  
        MongoDBDatabase spitdb = client.getDatabase("spitdb");//打开数据库  
        MongoCollection<Document> spit = spitdb.getCollection("spit");//  
        获取集合  
  
        Map<String,Object> map=new HashMap();  
        map.put("content","我要吐槽");  
        map.put("userid","9999");  
        map.put("visits",123);  
        map.put("publishtime",new Date());  
        Document document=new Document(map);  
  
        spit.insertOne(document);//插入数据  
        client.close();  
    }  
}
```

## 3.2 SpringDataMongoDB

SpringData家族成员之一，用于操作MongoDb的持久层框架，封装了底层的mongodb-driver。

官网主页：<https://projects.spring.io/spring-data-mongodb/>

我们十次方项目的吐槽微服务就采用SpringDataMongoDB框架

# 4 吐槽微服务

## 4.1 需求分析

采用SpringDataMongoDB框架实现吐槽微服务的持久层。

实现功能：

（1）基本增删改查API

- (2) 根据上级ID查询吐槽列表
- (3) 吐槽点赞
- (4) 发布吐槽

## 4.2 代码编写

### 4.2.1 模块搭建

- (1) 搭建子模块 tensquare\_spit
- (2) pom.xml引入依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
<dependency>
    <groupId>com.tensquare</groupId>
    <artifactId>tensquare_common</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```

- (3) 创建application.yml

```
server:
  port: 9006
spring:
  application:
    name: tensquare-spit #指定服务名
  data:
    mongodb:
      host: 192.168.184.134
      database: spitdb
```

- (4) 创建启动类



```
@SpringBootApplication
public class SpitApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpitApplication.class, args);
    }

    @Bean
    public IdWorker idWorker(){
        return new IdWorker(1, 1);
    }
}
```

## 4.2.2基本增删改查API实现

### (1) 创建实体类

创建包com.tensquare.spit，包下建包pojo 用于存放实体类，创建实体类



```
/**
 * 吐槽
 * @author Administrator
 *
 */
public class Spit implements Serializable{

    @Id
    private String _id;
    private String content;
    private Date publishtime;
    private String userid;
    private String nickname;
    private Integer visits;
    private Integer thumbup;
    private Integer share;
    private Integer comment;
    private String state;
    private String parentid;
    // getter and setter .....
}
```

## (2) 创建数据访问接口

com.tensquare.spit包下创建dao包，包下创建接口

```
/**
 * 吐槽数据访问层
 * @author Administrator
 *
 */
public interface SpitDao extends MongoRepository<Spit, String>{

}
```

## (3) 创建业务逻辑类

com.tensquare.spit包下创建service包，包下创建类



```
@Service
public class SpitService {

    @Autowired
    private SpitDao spitDao;

    @Autowired
    private IdWorker idWorker;

    /**
     * 查询全部记录
     * @return
     */
    public List<Spit> findAll(){
        return spitDao.findAll();
    }

    /**
     * 根据主键查询实体
     * @param id
     * @return
     */
    public Spit findById(String id){
        Spit spit = spitDao.findById(id).get();
        return spit;
    }

    /**
     * 增加
     * @param spit
     */
    public void add(Spit spit) {
        spit.set_id(idWorker.nextId()+""); //主键值
        spitDao.save(spit);
    }

    /**
     * 修改
     * @param spit
     */
}
```



```
public void update(Spit spit) {  
    spitDao.save(spit);  
}  
  
/**  
 * 删除  
 * @param id  
 */  
public void deleteById(String id) {  
    spitDao.deleteById(id);  
}  
}
```

(4) com.tensquare.spit包下创建controller类



```
@RestController
@CrossOrigin
@RequestMapping("/spit")
public class SpitController {

    @Autowired
    private SpitService spitService;

    /**
     * 查询全部数据
     * @return
     */
    @RequestMapping(method= RequestMethod.GET)
    public Result findAll(){
        return new Result(true, StatusCode.OK, "查询成功",spitService.findAll());
    }

    /**
     * 根据ID查询
     * @param id ID
     * @return
     */
    @RequestMapping(value="/{id}",method=RequestMethod.GET)
    public Result findOne(@PathVariable String id){
        return new Result(true,StatusCode.OK, "查询成功",spitService.findById(id));
    }

    /**
     * 增加
     * @param spit
     */
    @RequestMapping(method=RequestMethod.POST)
    public Result add(@RequestBody Spit spit ){
        spitService.add(spit);
        return new Result(true,StatusCode.OK, "增加成功");
    }

    /**
```



```
* 修改
* @param spit
*/
@RequestMapping(value="/{id}",method=RequestMethod.PUT)
public Result update(@RequestBody Spit spit,@PathVariable String id )
{
    spit.set_id(id);
    spitService.update(spit);
    return new Result(true,StatusCode.OK,"修改成功");
}

/**
* 删除
* @param id
*/
@RequestMapping(value="/{id}",method=RequestMethod.DELETE)
public Result deleteById(@PathVariable String id ){
    spitService.deleteById(id);
    return new Result(true,StatusCode.OK,"删除成功");
}
}
```

## 4.2.3 根据上级ID查询吐槽列表

### (1) SpitDao新增方法定义

```
/**
* 根据上级ID查询吐槽列表（分页）
* @param parentid
* @param pageable
* @return
*/
public Page<Spit> findByParentid(String parentid,Pageable pageable);
```

### (2) SpitService新增方法

```
/**
 * 根据上级ID查询吐槽列表
 * @param parentid
 * @param page
 * @param size
 * @return
 */
public Page<Spit> findByParentid(String parentid,int page, int size){
    PageRequest pageRequest = PageRequest.of(page-1, size);
    return spitDao.findByParentid(parentid, pageRequest);
}
```

### (3) SpitController新增方法

```
/**
 * 根据上级ID查询吐槽分页数据
 * @param page
 * @param size
 * @return
 */

@RequestMapping(value="/comment/{parentId}/{page}/{size}",method=RequestMethod.GET)
public Result findByParentid(@PathVariable String parentId,
    @PathVariable int page,@PathVariable int size){
    Page<Spit> pageList = spitService.findByParentid(parentId,page,
size);
    return new Result(true,StatusCode.OK,"查询成功",new
PageResult<Spit>(pageList.getTotalElements(), pageList.getContent() ) );
}
```

## 4.2.4 吐槽点赞

我们看一下以下点赞的代码： SpitService 新增updateThumbup方法



```
/**
 * 点赞
 * @param id
 */
public void updateThumbup(String id){
    Spit spit = spitDao.findById(id).get();
    spit.setThumbup(spit.getThumbup()+1);
    spitDao.save(spit);
}
```

以上方法虽然实现起来比较简单，但是执行效率并不高，因为我只需要将点赞数加1就可以了，没必要查询出所有字段修改后再更新所有字段。

我们可以使用MongoTemplate类来实现对某列的操作。

### (1) 修改SpitService

```
@Autowired
private MongoTemplate mongoTemplate;

/**
 * 点赞
 * @param id
 */
public void updateThumbup(String id){
    Query query=new Query();
    query.addCriteria(Criteria.where("_id").is(id));
    Update update=new Update();
    update.inc("thumbup",1);
    mongoTemplate.updateFirst(query,update,"spit");
}
```

### (2) SpitController新增方法



```
/**
 * 点赞
 * @param id
 * @return
 */
@RequestMapping(value="/thumbup/{id}",method=RequestMethod.PUT)
public Result updateThumbup(@PathVariable String id){
    spitService.updateThumbup(id);
    return new Result(true,StatusCode.OK,"点赞成功");
}
```

## 4.2.5 控制不能重复点赞

我们可以通过redis控制用户不能重复点赞

(1) 首先引入依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

(2) 修改application.yml

```
redis:
    host: 192.168.184.135
```

(3) 修改SpitController代码逻辑

```
@Autowired
private RedisTemplate redisTemplate;

/**
 * 吐槽点赞
 * @param id
 * @return
 */
@RequestMapping(value = "/thumbup/{id}", method = RequestMethod.PUT)
public Result updateThumbup(@PathVariable String id){
    //判断用户是否点过赞
    String userid="2023";// 后边我们会修改为当前登陆的用户
    if(redisTemplate.opsForValue().get("thumbup_"+userid+"_"+
id)!=null){
        return new Result(false,StatusCode.REPERROR,"你已经点过赞了");
    }

    spitService.updateThumbup(id);
    redisTemplate.opsForValue().set( "thumbup_"+userid+"_"+ id,"1");
    return new Result(true,StatusCode.OK,"点赞成功");
}
```

## 4.2.6 发布吐槽

修改SpitService的add方法

```
/**
 * 发布吐槽（或吐槽评论）
 * @param spit
 */
public void add(Spit spit){
    spit.set_id( idWorker.nextId()+" ");
    spit.setPublishtime(new Date()); //发布日期
    spit.setVisits(0); //浏览量
    spit.setShare(0); //分享数
    spit.setThumbup(0); //点赞数
    spit.setComment(0); //回复数
    spit.setState("1"); //状态
    if(spit.getParentid()!=null && !"".equals(spit.getParentid())){//
        如果存在上级ID, 评论
        Query query=new Query();

        query.addCriteria(Criteria.where("_id").is(spit.getParentid()));
        Update update=new Update();
        update.inc("comment",1);
        mongoTemplate.updateFirst(query,update,"spit");
    }
    spitDao.save(spit);
}
```

## 4.2.7 增加浏览量与分享数

学员实现

# 5 文章评论功能开发

## 5.1 表结构分析

集合结构：

专栏文章评论	comment		
字段名称	字段含义	字段类型	备注
_id	ID	文本	
articleid	文章ID	文本	
content	评论内容	文本	
userid	评论人ID	文本	
parentid	评论ID	文本	如果为0表示文章的顶级评论
publishdate	评论日期	日期	

## 5.2 代码编写

以下功能学员实现

### 4.2.1 新增评论

(1) 修改tensquare\_article工程的pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

(2) 修改application.yml,在spring节点下新增配置

```
data:
  mongodb:
    host: 192.168.184.134
    database: recruitdb
```

(3) 创建实体类



```
/**
 * 文章评论 (mongoDB)
 * @author Administrator
 *
 */
public class Comment implements Serializable{

    @Id
    private String _id;
    private String articleid;
    private String content;
    private String userid;
    private String parentid;
    private Date publishdate;

    //getter and setter....
}
```

#### (4) 创建数据访问接口

```
/**
 * 评论Dao
 * @author Administrator
 *
 */
public interface CommentDao extends MongoRepository<Comment, String> {

}
```

#### (5) 创建业务逻辑类

```
@Service
public class CommentService {

    @Autowired
    private CommentDao commentDao;

    @Autowired
    private IdWorker idWorker;

    public void add(Comment comment){
        comment.setId( idWorker.nextId()+"");
        commentDao.save(comment);
    }

}
```

#### (6) 创建控制器类

```
@RestController
@CrossOrigin
@RequestMapping("/comment")
public class CommentController {

    @Autowired
    private CommentService commentService;

    @RequestMapping(method= RequestMethod.POST)
    public Result save(@RequestBody Comment comment){
        commentService.add(comment);
        return new Result(true, StatusCode.OK, "提交成功 ");
    }

}
```

## 4.2.2 根据文章ID查询评论列表

#### (1) CommentDao新增方法定义

```
/**
 * 根据文章ID查询评论列表
 * @param articleid
 * @return
 */
public List<Comment> findByArticleid(String articleid);
```

## (2) CommentService新增方法

```
public List<Comment> findByArticleid(String articleid){
    return commentDao.findByArticleid(articleid);
}
```

## (3) CommentController新增方法

```
@RequestMapping(value="/article/{articleid}",method= RequestMethod.GET)
public Result findByArticleid(@PathVariable String articleid){
    return new Result(true, StatusCode.OK, "查询成功",
        commentService.findByArticleid(articleid));
}
```

## 4.2.3 删除评论

代码略

# 面试问题总结

你在项目中有没有使用到**mongodb**?

你的工程是**如何操作MongoDB**的?

spring data mongodb

在项目的哪些场景下使用**MongoDB**?

**吐槽**、文章评论

# 为什么在吐槽和文章评论中使用**Mongodb**而不使用**mysql**?

吐槽和评论都是数据量较大且价值较低的数据，为了减轻mysql的压力，我们使用  
mongodb