1. MongoDB
2. 为什么使用MonoDB

对于文章评论而言，数据量很大，同时有频繁地写入操作，且数据价值不高。鉴于数据量大这个特点，不适用于Redis这种缓存数据库，Redis适合快速读写的数据，而不适合大数据。Mysql虽然可以通过搭建集群来用于大数据，但是由于评论数据价值不高，用Mysql搭建集群性价比非常低。

因此，这里我们使用MongoDB来存储评论数据。

1. MongoDB介绍

MongoDB是一个基于分布式文件存储的数据库，介于关系型数据库和非关系型数据库之间，是非关系型数据库中功能最丰富的的，它支持比较松散的数据结构，因此可以存储比较复杂的数据类型，类似于Json的bson格式。

1. MongoDB特点

支持的查询语言很强大，语法类似于面向对象的查询语言，可以实现关系型数据库单表查询的绝大部分功能，还支持对数据建立索引。高性能，易部署，易使用，存储数据很方便。

1 . 面向集合存储，易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. 支持查询。

6. 支持复制和故障恢复。

7. 使用高效的二进制数据存储，包括大型对象（如视频等）。

8. 自动处理碎片，以支持云计算层次的扩展性。

9. 支持RUBY，PYTHON，JAVA，C++，PHP，C#等多种语言。

10. 文件存储格式为BSON（一种JSON的扩展）。

1. MongoDB层次结构

MongoDB的逻辑结构是由三部分组成，分别是文档，集合和数据库。文档就相当于数据表中的一行数据，集合相当于一个数据表，数据库还是数据库。一个数据库中有多个集合，一个集合中有多个文档。一个MongoDB实例支持多个数据库。

1. MongoDB数据类型
2. ObjectId

一个文档唯一的标识，类似于主键，可以很快地生成和排序，一共12bytes，前4各个字节表示创建UNIX时间戳，UTC时间。接下来的3个字节是机器标识码，接着的两个字节是进程ID，最后三个字节是随机数。

MongoDB中存储的文档必须有一个\_id键，这个键的值可以是任意类型的，如果不手动指定id，默认是个ObjectId对象。推荐使用默认id，因为指定id会减慢插入的速度。

1. 时间戳

特殊的时间戳，与普通的日期类型不相关，64位，前32位是一个time\_t值，与1970年1月1日相差的秒数；后32位是在某秒中操作的一个递增的序数。

1. 日期

表示当前时间距离1970年1月1日的毫秒数，如果是1970年之前的就是负数。

1. MongoDB的安装与使用
2. Windows

通常，我们需要在Windows的MongoDB客户端中连接服务器，因此也需要在Windows中安装MongoDB。

1. 启动

在安装目录下的bin目录下有mongod.exe，这是启动服务的，还有一个mongo.exe，这是客户端用于连接服务。在该目录下打开cmd，输入以下的命令，启动服务，后面的路径是数据存放的位置：

mongod –dbpath=D:\MR.WANG\JAVA\tensquare\data

可以看到出现了默认端口号27017，说明启动成功，如果使用默认端口号27017，可以不用输入端口号。

1. 连接

同样的，在cmd中输入mongo命令，可以省略端口参数-port，因为是默认端口号：

mongo

出现MongoDB的版本说明连接成功。

要使用其他的端口号，在mongo命令后面添加主机:端口

mongo 127.0.0.1:8080

1. Linux中的Docker环境

在Docker中创建mongo容器，将宿主机的27017端口映射到Docker容器的27017端口中，那么我们就可以在本地通过连接宿主机和Docker中的mongo容器通信了。

1. MongoDB的常用命令
2. 选择和创建数据库

如果数据库存在则选择，不存在则创建

use 数据库名称

1. 查看数据库

show dbs

1. 查看集合

首先选择数据库，才能查看该数据库的集合：

show collections

1. 插入文档

使用集合来进行插入文档

db.集合名称.insert(数据)

注意，这里插入的数据都是bson格式的，语法和json是一样的，都是键值对。键值都是随意指定的，同时在文档中有一个键值为”\_id”，如果你不指定的话，它会默认生成一个，类型是ObjectId类型的，自己指定的话可以是MongDB支持的任意类型。

1. 查询集合
2. 查询所有的文档

db.集合名称.find()

1. 有条件地查询文档

通过字段名称来查询，find中的依然是bson格式的：

db.comment.find({“userid”:”111”})

1. findOne

返回符合条件的第一条数据

1. limit()

返回指定条数的记录，可以在find()方法后面调用limit来返回结果：

db.comment.find().limit(2)

1. 修改与删除文档
2. update

update用于修改文档，但是它是先将该条文档删除，再重新写入新的文档，这样，并不合理，有些不想修改的字段也会被修改掉。

db.comment.update({\_id:”1”}, {thumbup:1000})

1. 使用修改器$set

使用修改器能够指定修改的键值对，不会对其他的键值对造成影响。如下所示：

db.comment.update({\_id:”2”}, {$set:{thumbup:2000}})

1. 有条件地删除文档

同样的，在括号中添加条件，如某个键值对：

db.comment.remove({thumbup:1000})

1. 删除所有文档

即没有条件，直接删除

db.comment.remove({})

1. 统计条数

分为有条件地统计文档数和统计所有的文档数，用于与删除一样：

db.comment.count()

db.comment.count({userId:”1013”})

1. 模糊查询

MongoDB的模糊查询是通过正则表达式来实现的，格式如下：

/查询字符串/

查询字符串就是正则表达式的查询字段。

1. 查询content字段包含aa字符的文档

db.comment.find({content:/aa/})

1. 查询以aa开头的content字段的文档

db.comment.find({content:/^aa/})

1. 数字的比较

$gt 表示 大于， $lt 表示小于， $gte 表示大于等于， $lte 表示小于等于， $ne 表示不等于。如下所示：

db.comment.find({“field”:{$gt:value}})

该表达式表示在comment集合中查找有无文档的field字段是大于value的。

1. 包含与不包含

查询某个字段是否包含某些字符，如下所示：

db.comment.find({userId:{$in:[“10”, “12”]}})

表示查询comment集合中的userId字段是否包含有10 或者 12.

同理，$nin 表示不包含。

（十一）条件连接

1. $and

与条件，即并且关系。

$and:[ {条件1}, {条件2}，{条件3} ]

db.comment.find({ $and:[ {thumbup:{$gte:1000}}, {thumbup:{$lt:2000}}]})

1. $or

或关系，同理。

（十二） 增长字段值

$inc单独对某个字段值进行增加或者减少，设置正数就是增加，设置负数就是减少。也是使用update方法。

db.comment.update({\_id:”2”}, {$inc:{thumbup:1}})

1. mongodb-driver的使用

mongodb-driver是mongo官方推出的java连接mongoDB的驱动包。

1. 添加依赖

依然是在tensquare\_parent父工程下创建子工程mongodb\_demo。添加mongodb-driver的依赖。

<dependencies>  
 <dependency>  
 <groupId>org.mongodb</groupId>  
 <artifactId>mongodb-driver</artifactId>  
 <version>3.10.1</version>  
 </dependency>  
</dependencies>

1. 查询所有文档

使用mongodb-driver查询文档的步骤和在mongodb客户端中是一样的。

1. 创建操作MongoDB的客户端

MongoClient mongoClient = new MongoClient("192.168.112.128");

1. 选择数据库

MongoDatabase commentdb = mongoClient.getDatabase("commentdb");

1. 获取集合

MongoCollection<Document> comment = commentdb.getCollection("comment");

1. 使用集合进行查询

FindIterable<Document> documents = comment.find();

1. 解析结果集

for(Document document : documents){  
 System.*out*.println("----------");  
 System.*out*.println("\_id" + document.get("\_id"));  
}

1. 释放资源，关闭客户端

mongoClient.close();

1. 改造测试

使用Junit测试时，可以使用@Before注解表示在@Test注解的方法执行前执行，通常用于创建资源，注解@After用于在注解@Test标注的方法后执行，通常用于资源的释放。

因此，这里把MongoDB的客户端，创建的数据库，集合都放在注解@Before下面的方法中实现，把客户端资源的释放放在注解@After中的方法下。

完整代码如下所示：

import com.mongodb.MongoClient;  
import com.mongodb.client.FindIterable;  
import com.mongodb.client.MongoCollection;  
import com.mongodb.client.MongoDatabase;  
import org.bson.Document;  
import org.junit.After;  
import org.junit.Before;  
import org.junit.Test;  
  
public class mongodbTest {  
  
 private MongoClient mongoClient;  
 private MongoCollection<Document> comment;  
  
 @Before  
 public void init(){  
 mongoClient = new MongoClient("192.168.112.128");  
 MongoDatabase commentdb = mongoClient.getDatabase("commentdb");  
 comment = commentdb.getCollection("comment");  
 }  
  
 @Test  
 public void test1(){  
 FindIterable<Document> documents = comment.find();  
 for(Document document : documents){  
 System.*out*.println("----------");  
 System.*out*.println("\_id" + document.get("\_id"));  
 }  
 }  
  
 @After  
 public void after(){  
 mongoClient.close();  
 }  
}

1. 条件查询
2. 创建查询条件bson

像构造json格式数据一样构造bson数据：

BasicDBObject bson = new BasicDBObject("\_id", "1");

1. 执行查询

FindIterable<Document> documents = comment.find(bson);  
for(Document document : documents){  
 System.*out*.println("\_id: " + document.get("\_id"));  
}

1. 插入单条文档
2. 通过map封装数据

Map<String, Object> map = new HashMap<>();  
map.put("\_id", "6");  
map.put("content", "test");  
map.put("userid", "11");  
map.put("thumbup", "22");

1. 封装新增文档对象

Document document = new Document(map);

1. 执行新增

comment.insertOne(document);

1. 修改文档

修改文档需要提供修改的条件，即在满足什么条件下才能修改，同时还要提供修改的键值对的内容。修改的条件和修改的内容都是使用的bson格式的数据，首先需要创建一个修改的条件bson，再创建一个需要修改的文档中的键值对bson，再使用updateOne传入两个bson，即可执行修改。

public void test4(){  
 BasicDBObject filter = new BasicDBObject("\_id", "6");  
 BasicDBObject update = new BasicDBObject("$set", new Document("userid", "11"));  
 comment.updateOne(filter, update);  
}

1. 删除文档

同样是创建一个删除条件的bson，传入到deleteOne函数中，执行删除。

public void test5(){  
 BasicDBObject filter = new BasicDBObject("\_id", "6");  
 comment.deleteOne(filter);  
}

1. 文章评论的实现
2. 需求分析

每篇文章下都会有评论，且每个评论下也可能会有其他人新的评价，因此，需要在评论的数据库中添加评论父ID，表示该评论所针对的对象。同时还有该评论所处的文章ID，评论内容，评论人ID，评论日期，点赞数量。

要实现的功能是增删改查，根据文章ID查询评论，给评论点赞。

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 评论父ID 文本 如果为0表示文章的顶级评论

publishdate 评论日期 日期

thumbup 点赞数 数字

1. 技术实现

使用SpringDataMongoDB，它是用于操作MongoDB的持久层框架，封装了底层的mongodb-driver。

1. 添加依赖，配置mongodb

在artcile子项目中，添加data-mongodb依赖：

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

配置mongodb

spring:

data:  
 mongodb:  
 database: commentdb  
 host: 192.168.112.128

1. 封装评论POJO

将设计好的评论结构封装称为POJO，

package com.tensquare.article.pojo;  
  
import org.springframework.data.annotation.Id;  
  
import java.io.Serializable;  
import java.util.Date;  
  
public class Comment implements Serializable {  
  
 @Id  
 private String \_id;  
 private String articleid;  
 private String content;  
 private String userid;  
 private String parentid;  
 private Date publishdate;  
 private Integer thumbup;  
  
 public String get\_id() {  
 return \_id;  
 }  
  
 public void set\_id(String \_id) {  
 this.\_id = \_id;  
 }  
  
 public String getArticleid() {  
 return articleid;  
 }  
  
 public void setArticleid(String articleid) {  
 this.articleid = articleid;  
 }  
  
 public String getContent() {  
 return content;  
 }  
  
 public void setContent(String content) {  
 this.content = content;  
 }  
  
 public String getUserid() {  
 return userid;  
 }  
  
 public void setUserid(String userid) {  
 this.userid = userid;  
 }  
  
 public String getParentid() {  
 return parentid;  
 }  
  
 public void setParentid(String parentid) {  
 this.parentid = parentid;  
 }  
  
 public Date getPublishdate() {  
 return publishdate;  
 }  
  
 public void setPublishdate(Date publishdate) {  
 this.publishdate = publishdate;  
 }  
  
 public Integer getThumbup() {  
 return thumbup;  
 }  
  
 public void setThumbup(Integer thumbup) {  
 this.thumbup = thumbup;  
 }  
  
 @Override  
 public String toString() {  
 return "Comment{" +  
 "\_id='" + \_id + '\'' +  
 ", articleid='" + articleid + '\'' +  
 ", content='" + content + '\'' +  
 ", userid='" + userid + '\'' +  
 ", parentid='" + parentid + '\'' +  
 ", publishdate=" + publishdate +  
 ", thumbup=" + thumbup +  
 '}';  
 }  
}

1. 持久层repository编写

注意，操作MongoDB的类需要创建一个新的包repository，不要与原有的Mybatsi的接口放在一个包中。编写接口的方法与Mybasis差不多，只需要继承MongoRepository接口后，就可以使用该接口中的许多方法。注意，要在该接口中声明操作的对象以及对象的主键类型，如下所示：

package com.tensquare.article.repository;  
  
import com.tensquare.article.pojo.Comment;  
import org.springframework.data.mongodb.repository.MongoRepository;  
  
public interface CommentRepository extends MongoRepository<Comment, String> {  
}

1. Service层编写

package com.tensquare.article.service;  
  
import com.tensquare.article.repository.CommentRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
@Service  
public class CommentService {  
 @Autowired  
 private CommentRepository commentRepository;  
}

1. Controller层编写

package com.tensquare.article.controller;  
  
import com.tensquare.article.service.CommentService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
@RequestMapping("comment")  
public class CommentController {  
   
 @Autowired  
 private CommentService commentService;  
}

1. 实现查询所有评论功能
2. 编写 Controller层的findAll()方法

//GET /comment 查询所有评论  
@RequestMapping(method= RequestMethod.*GET*)  
public Result findAll(){  
 List<Comment> list = commentService.findAll();  
  
 return new Result(true, StatusCode.*OK*, "查询成功", list);  
}

1. 编写Service层的findAll()方法

直接调用继承了MongoRepository接口的持久层的方法findAll()，查询出所有的评论。

public List<Comment> findAll() {  
 List<Comment> list = commentRepository.findAll();  
 return list;  
}

1. 根据评论id查询评论数据
2. 编写Controller的findById()

//GET /comment/{commentId} 根据评论id查询评论数据  
@RequestMapping(value="{commentId}", method=RequestMethod.*GET*)  
public Result findById(@PathVariable String commentId){  
 Comment comment = commentService.findById(commentId);  
 return new Result(true, StatusCode.*OK*, "查询成功", comment);  
}

2 编写Service层的findById()方法

同样是调用接口findById即可，但是注意这里findById返回的是一个Optional，需要使用get函数获取Optional中的值。get函数的本意是为了防止空指针，只有非空指针才能通过get函数获取值。但是这样当传入的值真的是空指针时，就会抛出异常，我们在调用get函数时可以首先做一个判断，判断optional中是否为空，不为空就使用get函数。

public Comment findById(String commentId) {  
 Optional<Comment> optional = commentRepository.findById(commentId);  
 if(optional.isPresent()){  
 return optional.get();  
 }  
 return null;  
}

1. 新增评论功能
2. 编写Controller层方法save

新增评论功能需要我们传入一个json格式数据，转换为Comment对象，再将该对象存储到数据库中。

//POST /comment 新增评论  
@RequestMapping(method=RequestMethod.*POST*)  
public Result save(@RequestBody Comment comment){  
 commentService.save(comment);  
 return new Result(true, StatusCode.*OK*, "新增成功");  
}

1. 编写Service层方法save

首先，我们要保证评论id的唯一性，使用分布式id生成器生成id。同时要初始化点赞数据，发布时间，最后要保存数据。

public void save(Comment comment) {  
 String id = idWorker.nextId() + "";  
 comment.set\_id(id);  
  
 comment.setThumbup(0);  
 comment.setPublishdate(new Date());  
  
 commentRepository.save(comment);  
}

注意，在数据量比较大的时候，我们最好不要直接设定id，因为自己设定id会使数据查询的效率降低。让系统自动地生成id，再创建一个c\_id字段，用于存储分布式id生成器生成的id，这样既能保证查询的高效，也能保证id的唯一。

（十一） 修改评论功能

1. Controller层updateById方法

传入commentId和要修改的内容comment。

//PUT /comment/{commentId} 修改评论  
@RequestMapping(value="{commentId}", method = RequestMethod.*PUT*)  
public Result updateById(@PathVariable String commentId, @RequestBody Comment comment){  
 comment.set\_id(commentId);  
 commentService.updateById(comment);  
 return new Result(true, StatusCode.*OK*, "修改成功");  
}

1. Service层updateById方法

这里使用的是MongoRepository的save方法，该方法会判断如果主键存在则修改，如果不存在则插入。

public void updateById(Comment comment) {  
 commentRepository.save(comment);  
}

（十二） 根据id删除评论功能

1. Controller层的deleteById方法

//DELETE /comment/{commentId} 根据id删除评论  
@RequestMapping(value="{commentId}", method=RequestMethod.*DELETE*)  
public Result deleteById(@PathVariable String commentId){  
 commentService.deleteById(commentId);  
 return new Result(true, StatusCode.*OK*, "删除成功")  
}

1. Service层的deleteById方法

public void deleteById(String commentId) {  
 commentRepository.deleteById(commentId);  
}

（十三） 根据文章ID查询文章评论

1. Controller层findByArticleId方法

//GET /comment/article/{articleId} 根据文章Id查询文章评论  
@RequestMapping(value="article/{articleId}", method=RequestMethod.*GET*)  
public Result findByArticleId(@PathVariable String articleId){  
 List<Comment> list = commentService.findByArticleId(articleId);  
 return new Result(true, StatusCode.*OK*, "查询成功", list);  
}

1. Service层的findByArticleId方法

当我们需要自定义查询方式时，通常的方法是自己编写方法，有没有简便的方式呢？

有的，SpringDataMongoDB支持通过查询方法名来进行查询定义的方式，什么叫做查询方法名呢？就是创建一个方法，该方法的名称就是查询的语义。比如说我们要通过文章id来查询，那么该方法就可以定义为

List<Comment> findByArticleId();

ArticleId是Comment对象中的一个属性，因此它才能够识别出来方法名中的属性名称,因此首先需要添加返回的Comment对象列表，才能够识别方法名。

再如，根据用户id查询，并且根据发布时间倒序排序，方法可以定义为如下：

List<Comment> findByUseridOrderByPublishDateDesc();

我们只需要在持久层的接口中声明这些自定义的方法，MongoRepository会帮我们进行查询，我们在Service层中可以直接调用这些自定义的方法。

因此，这里可以在持久层CommentRepository接口中定义相应的查询方法，根据文章id查询文章评论数据：

List<Comment> findByArticleid(String articleId);

然后在Service层中，编写该方法，调用刚刚定义在接口commentReposory中的findByArticleid方法：

public List<Comment> findByArticleId(String articleId) {  
 List<Comment> list = commentRepository.findByArticleid(articleId);  
 return list;  
}

（十四） 点赞评论

1. Controller层的thumbup方法

//PUT /comment/thumbup/{commentId} 根据评论id点赞评论  
@RequestMapping(value="thumbup/{commentId}", method = RequestMethod.*PUT*)  
public Result thumbup(@PathVariable String commentId){  
 commentService.thumbup(commentId);  
 return new Result(true, StatusCode.*OK*, "点赞成功");  
}

1. Service层的thumbup方法

public void thumbup(String commentId) {  
 Comment comment = commentRepository.findById(commentId).get();  
 comment.setThumbup(comment.getThumbup() + 1);  
 commentRepository.save(comment);  
}

1. Service层thumbup方法优化

上面的thumbup方法是非线程安全的，我们可以使用Mongdb的inc方法来增加字段值，mongodb自身就是线程安全的，因此最好使用inc方法。首先，注入一个MongoTemplate模板类，使用该类的updateFirst方法，需要一个查询条件Query，一个封装修改的数值的Update对象，还有一个要修改的集合名称。

public void thumbup(String commentId) {  
// Comment comment = commentRepository.findById(commentId).get();  
// comment.setThumbup(comment.getThumbup() + 1);  
// commentRepository.save(comment);  
 //封装修改的条件  
 Query query = new Query();  
 query.addCriteria(Criteria.*where*("\_id").is(commentId));  
  
 //封装修改的数据  
 Update update = new Update();  
 update.inc("thumbup", 1);  
  
 mongoTemplate.updateFirst(query, update, "comment");  
 }

1. 禁止重复点赞

一个用户不能重复点赞，可以在用户点完赞后添加标记，在每次用户点赞之前检查该标记。这里我们使用redis保存点赞信息，在Redis通过一个标记，userId + “\_” + commentId拼接成一个字段名称，用于记录该用户是否点赞了该评论。

首先，在docker中创建一个redis容器；

在article子项目的pom.xml中添加redis依赖：

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

然后在配置文件中配置redis的主机，端口：

spring:

redis:  
 host: 192.168.112.128

修改Controller中的thumbup方法，首先从Redis中查询标记信息，如果为空，说明该用户没有点赞可以进行点赞，否则不能点赞。

//PUT /comment/thumbup/{commentId} 根据评论id点赞评论  
@RequestMapping(value="thumbup/{commentId}", method = RequestMethod.*PUT*)  
public Result thumbup(@PathVariable String commentId){  
 //将用户点赞信息保存到Redis中  
 //模拟用户id  
 String userId = "123";  
  
 //查询用户点赞信息，根据用户id和评论id  
 Object flag = redisTemplate.opsForValue().get("thumbup\_" + userId + "\_" + commentId);  
 //判断查询到的结果是否为空  
 if(flag == null){  
 //为空，说明用户没有点赞  
 commentService.thumbup(commentId);  
 redisTemplate.opsForValue().set("thumbup\_" + userId + "\_" + commentId, 1);  
 return new Result(true, StatusCode.*OK*, "点赞成功");  
 }else{  
 return new Result(false, StatusCode.*REPERROR*, "不能重复点赞");  
 }  
 //commentService.thumbup(commentId);  
 //return new Result(true, StatusCode.OK, "点赞成功");  
}