课程说明

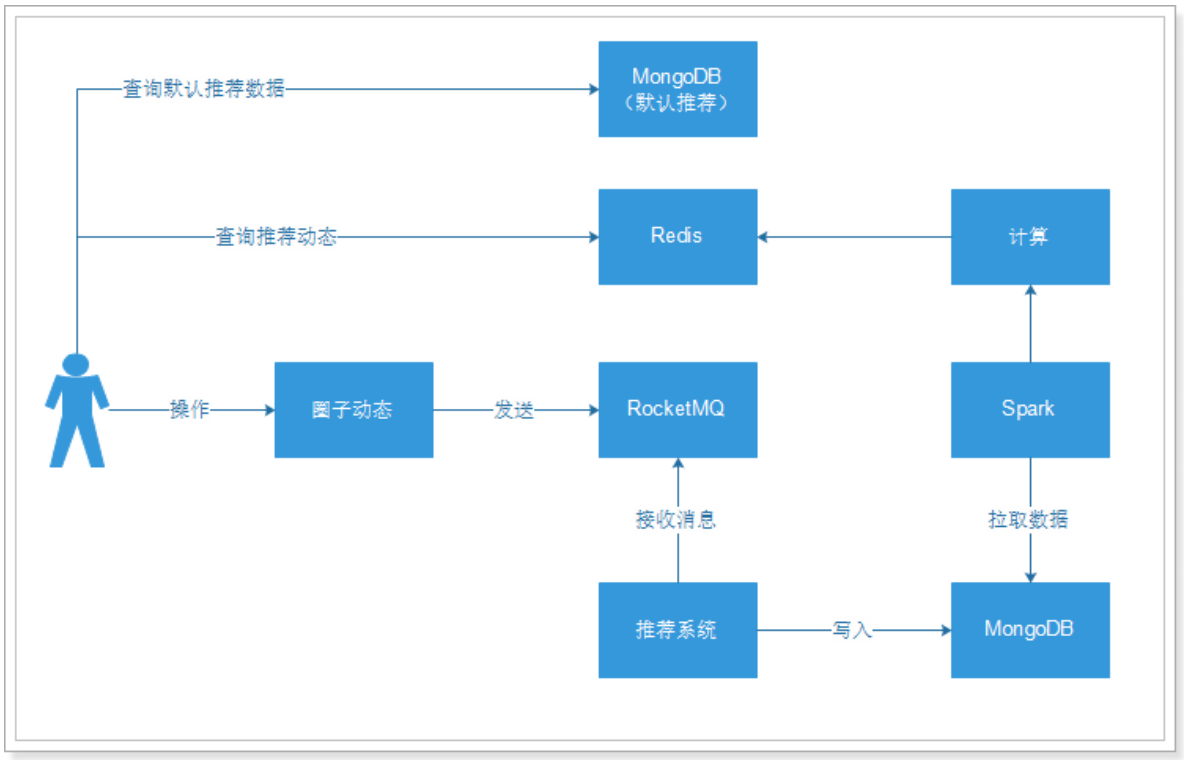
- 圈子推荐功能说明
- 圈子推荐功能流程
- 圈子推荐功能的实现
- 搭建推荐系统
- 使用Spark实现推荐逻辑
- 小视频推荐功能的实现

1、圈子推荐

1.1、功能说明

在圈子功能中，针对于用户发布的动态信息，系统可以根据用户的发布、浏览、点赞等操作，对动态信息做计算，然后对每个用于进行不同的推荐。

1.2、流程说明



流程说明：

- 用户对圈子的动态操作，如：发布、浏览、点赞、喜欢等，就会给RocketMQ进行发送消息；
- 推荐系统接收消息，并且处理消息数据，处理之后将结果数据写入到MongoDB中；
- Spark系统拉取数据，然后进行推荐计算；
- 计算之后的结果数据写入到Redis中，为每个用户都进行个性化推荐；
- 如果有用户没有数据的，查询MongoDB中的默认数据；

1.3、动态增加自增id

由于我们使用的推荐模型中，动态id需要是Long类型的，而我们之前使用的ObjectId类型的，所以需要增加Long类型的id。

1.3.1、修改Publish对象

```
package com.tanhua.dubbo.server.pojo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.bson.types.ObjectId;
import org.springframework.data.mongodb.core.mapping.Document;

import java.util.Date;
import java.util.List;

/**
 * 发布表，动态内容
 */
@Data
@NoArgsConstructor
@AllArgsConstructor
@Document(collection = "quanzi_publish")
public class Publish implements java.io.Serializable {

    private static final long serialVersionUID = 8732308321082804771L;

    private ObjectId id; //主键id
    private Long pid; //Long类型的id，用于推荐引擎使用
    private Long userId;
    private String text; //文字
    private List<String> medias; //媒体数据，图片或小视频 url
    private Integer seeType; // 谁可以看，1-公开，2-私密，3-部分可见，4-不给谁看
    private List<Long> seeList; //部分可见的列表
    private List<Long> notSeeList; //不给谁看的列表
    private String longitude; //经度
    private String latitude; //纬度
    private String locationName; //位置名称
    private Long created; //发布时间

}


```

1.3.2、修改发布逻辑

```
@Override
    public String savePublish(Publish publish) {

        // 校验
        if (publish.getUserId() == null) {
            return null;
        }

        try {
            publish.setCreated(System.currentTimeMillis()); //设置创建时间
```

```
            publish.setId(ObjectId.get()); //设置id
            publish.setPid(this.idService.createId("publish",
publish.getId().toHexString()));

            this.mongoTemplate.save(publish); //保存发布

            Album album = new Album(); // 构建相册对象
            album.setPublishId(publish.getId());
            album.setCreated(System.currentTimeMillis());
            album.setId(ObjectId.get());
            this.mongoTemplate.save(album, "quanzi_album_" +
publish.getUserId());

            //写入好友的时间线中
            Criteria criteria =
Criteria.where("userId").is(publish.getUserId());
            List<Users> users =
this.mongoTemplate.find(Query.query(criteria), Users.class);
            for (Users user : users) {
                TimeLine timeLine = new TimeLine();
                timeLine.setId(ObjectId.get());
                timeLine.setPublishId(publish.getId());
                timeLine.setUserId(user.getUserId());
                timeLine.setDate(System.currentTimeMillis());
                this.mongoTemplate.save(timeLine, "quanzi_time_line_" +
user.getFriendId());
            }

            return publish.getId().toHexString();
        } catch (Exception e) {
            e.printStackTrace();
            //TODO 出错的事务回滚
        }

        return null;
    }
```

```
package com.tanhua.dubbo.server.service;

import org.apache.commons.lang3.StringUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Service;

//生成自增长的id，原理：使用redis的自增长值
@Service
public class IdService {

    @Autowired
    private RedisTemplate<String, String> redisTemplate;

    public Long createId(String type, String strId) {
        type = StringUtils.upperCase(type);

        String idHashKey = "TANHUA_ID_HASH_" + type;
        if (this.redisTemplate.opsForHash().hasKey(idHashKey, strId)) {
```

```
            return
Long.valueOf(this.redisTemplate.opsForHash().get(idHashKey,
strId).toString());
        }

        String idKey = "TANHUA_ID_" + type;
        Long id = this.redisTemplate.opsForValue().increment(idKey);

        this.redisTemplate.opsForHash().put(idHashKey, strId,
id.toString());

        return id;
    }

}
```

itcast-tanhua-dubbo-service需要增加Redis依赖：

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

1.4、动态计分规则

- 浏览 +1
- 点赞 +5
- 喜欢 +8
- 评论 + 10
- 文字长度：50以内1分，50~100之间2分，100以上3分
- 图片个数：每个图片一分

1.5、发送消息

1.5.1、QuanziMQService

itcast-tanhua-server增加依赖：

```
<!--RocketMQ相关-->
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-spring-boot-starter</artifactId>
    <version>2.0.0</version>
</dependency>
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-client</artifactId>
    <version>4.3.2</version>
</dependency>
```

配置文件：

```
#rocketmq相关配置
spring.rocketmq.nameServer=192.168.31.81:9876
spring.rocketmq.producer.group=tanhua
```

```
package com.tanhua.server.service;

import com.alibaba.dubbo.config.annotation.Reference;
import com.tanhua.dubbo.server.api.QuanZiApi;
import com.tanhua.dubbo.server.pojo.Publish;
import com.tanhua.server.pojo.User;
import com.tanhua.server.utils.UserThreadLocal;
import org.apache.rocketmq.spring.core.RocketMQTemplate;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;

@Service
public class QuanziMQService {

    private static final Logger LOGGER =
LoggerFactory.getLogger(QuanziMQService.class);

    @Autowired
    private RocketMQTemplate rocketMQTemplate;

    @Reference(version = "1.0.0")
    private QuanZiApi quanZiApi;

    /**
     * 发布动态消息
     *
     * @param publishId
     * @return
     */
    public Boolean publishMsg(String publishId) {
        return this.sendMsg(publishId, 1);
    }

    /**
     * 浏览动态消息
     *
     * @param publishId
     * @return
     */
    public Boolean queryPublishMsg(String publishId) {
        return this.sendMsg(publishId, 2);
    }

    /**
```

```
50          * 点赞动态消息
51          *
52          * @param publishId
53          * @return
54          */
55         public Boolean likePublishMsg(String publishId) {
56             return this.sendMsg(publishId, 3);
57         }
58
59         /**
60          * 取消点赞动态消息
61          *
62          * @param publishId
63          * @return
64          */
65         public Boolean disLikePublishMsg(String publishId) {
66             return this.sendMsg(publishId, 6);
67         }
68
69         /**
70          * 喜欢动态消息
71          *
72          * @param publishId
73          * @return
74          */
75         public Boolean lovePublishMsg(String publishId) {
76             return this.sendMsg(publishId, 4);
77         }
78
79         /**
80          * 取消喜欢动态消息
81          *
82          * @param publishId
83          * @return
84          */
85         public Boolean disLovePublishMsg(String publishId) {
86             return this.sendMsg(publishId, 7);
87         }
88
89         /**
90          * 评论动态消息
91          *
92          * @param publishId
93          * @return
94          */
95         public Boolean commentPublishMsg(String publishId) {
96             return this.sendMsg(publishId, 5);
97         }
98
99         /**
100          * 发送圈子操作相关的消息
101          *
102          * @param publishId
103          * @param type      1-发动态，2-浏览动态， 3-点赞， 4-喜欢， 5-评论，6-取消点
     赞，7-取消喜欢
104          * @return
105          */
106         private Boolean sendMsg(String publishId, Integer type) {
```

```
        try {
            User user = UserThreadLocal.get();

            Publish publish = this.quanZiApi.queryPublishById(publishId);

            //构建消息
            Map<String, Object> msg = new HashMap<>();
            msg.put("userId", user.getId());
            msg.put("date", System.currentTimeMillis());
            msg.put("publishId", publishId);
            msg.put("pid", publish.getPid());
            msg.put("type", type);

            this.rocketMQTemplate.convertAndSend("tanhua-quanzi", msg);
        } catch (Exception e) {
            LOGGER.error("发送消息失败! publishId = " + publishId + ", type = " + type, e);
            return false;
        }

        return true;
    }
}

```

1.5.2、修改MovementsController

```
package com.tanhua.server.controller;

import com.tanhua.server.service.MovementsService;
import com.tanhua.server.service.QuanziMQService;
import com.tanhua.server.vo.Movements;
import com.tanhua.server.vo.PageResult;
import org.apache.commons.lang3.StringUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

@RestController
@RequestMapping("movements")
public class MovementsController {

    @Autowired
    private MovementsService movementsService;

    @Autowired
    private QuanziMQService quanziMQService;

    /**
     * 发送动态
     *
     * @param textContent
     * @param location
     * @param multipartFile
     * @return
```

```
     */
    @PostMapping()
    public ResponseEntity<Void> savePublish(@RequestParam("textContent") String textContent,
                                            @RequestParam("location") String location,
                                            @RequestParam("longitude") String longitude,
                                            @RequestParam("latitude") String latitude,
                                            @RequestParam(value = "imageContent", required = false) MultipartFile[] multipartFile) {
        try {
            String publishId = this.movementsService.savePublish(textContent, location, longitude, latitude, multipartFile);
            if (StringUtils.isNotEmpty(publishId)) {
                // 发送消息
                this.quanziMQService.publishMsg(publishId);
                return ResponseEntity.ok(null);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }

    /**
     * 查询好友动态
     *
     * @param page
     * @param pageSize
     * @return
     */
    @GetMapping
    public PageResult queryPublishList(@RequestParam(value = "page", defaultValue = "1") Integer page,
                                       @RequestParam(value = "pagesize", defaultValue = "10") Integer pageSize) {
        return this.movementsService.queryPublishList(page, pageSize, false);
    }

    /**
     * 查询推荐动态
     *
     * @param page
     * @param pageSize
     * @return
     */
    @GetMapping("recommend")
    public PageResult queryRecommendPublishList(@RequestParam(value = "page", defaultValue = "1") Integer page,
                                                @RequestParam(value = "pagesize", defaultValue = "10") Integer pageSize) {
        return this.movementsService.queryPublishList(page, pageSize, true);
```

```
75      }
76
77      /**
78       * 点赞
79       *
80       * @param publishId
81       * @return
82       */
83      @GetMapping("/{id}/like")
84      public ResponseEntity<Long> likeComment(@PathVariable("id") String
publishId) {
85          try {
86              Long likeCount = this.movementsService.likeComment(publishId);
87              if (likeCount != null) {
88
89                  //发送点赞消息
90                  this.quanziMQService.likePublishMsg(publishId);
91
92                  return ResponseEntity.ok(likeCount);
93              }
94          } catch (Exception e) {
95              e.printStackTrace();
96          }
97          return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
98      }
99
100     /**
101      * 取消点赞
102      *
103      * @param publishId
104      * @return
105      */
106     @GetMapping("/{id}/dislike")
107     public ResponseEntity<Long> disLikeComment(@PathVariable("id") String
publishId) {
108         try {
109             Long likeCount =
this.movementsService.cancelLikeComment(publishId);
110             if (null != likeCount) {
111
112                 //发送取消点赞消息
113                 this.quanziMQService.disLikePublishMsg(publishId);
114
115                 return ResponseEntity.ok(likeCount);
116             }
117         } catch (Exception e) {
118             e.printStackTrace();
119         }
120         return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
121     }
122
123     /**
124      * 喜欢
125      *
126      * @param publishId
127      * @return
```

```
     */
    @GetMapping("/{id}/love")
    public ResponseEntity<Long> loveComment(@PathVariable("id") String publishId) {
        try {
            Long loveCount = this.movementsService.loveComment(publishId);
            if (null != loveCount) {

                //发送喜欢消息
                this.quanziMQService.lovePublishMsg(publishId);

                return ResponseEntity.ok(loveCount);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }

    /**
     * 取消喜欢
     *
     * @param publishId
     * @return
     */
    @GetMapping("/{id}/unlove")
    public ResponseEntity<Long> disLoveComment(@PathVariable("id") String publishId) {
        try {
            Long loveCount = this.movementsService.cancelLoveComment(publishId);
            if (null != loveCount) {

                //发送取消喜欢消息
                this.quanziMQService.disLovePublishMsg(publishId);

                return ResponseEntity.ok(loveCount);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }

    /**
     * 查询单条动态信息
     *
     * @param publishId
     * @return
     */
    @GetMapping("/{id}")
    public ResponseEntity<Movements> queryById(@PathVariable("id") String publishId) {
        try {
            Movements movements = this.movementsService.queryById(publishId);
```

```
            if (null != movements) {

                //发送消息
                this.quanziMQService.queryPublishMsg(publishId);

                return ResponseEntity.ok(movements);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }

}

```

CommentsController：

```
 /**
     * 发表评论
     *
     * @param param
     * @return
     */
    @PostMapping
    public ResponseEntity<Void> saveComments(@RequestBody Map<String,
String> param) {
        try {
            String publishId = param.get("movementId");
            String content = param.get("comment");
            Boolean bool = this.commentsService.saveComments(publishId,
content);
            if (bool) {

                //发送消息
                this.quanziMQService.sendCommentPublishMsg(publishId);

                return ResponseEntity.ok(null);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }
```

1.6、接收消息

1.6.1、创建itcast-tanhua-recommend工程

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>itcast-tanhua</artifactId>
        <groupId>cn.itcast.tanhua</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>itcast-tanhua-recommend</artifactId>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-mongodb</artifactId>
        </dependency>
        <!--其他工具包依赖-->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
        </dependency>
        <dependency>
            <groupId>commons-io</groupId>
            <artifactId>commons-io</artifactId>
            <version>2.6</version>
        </dependency>
        <!--RocketMQ相关-->
        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-spring-boot-starter</artifactId>
            <version>2.0.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-client</artifactId>
            <version>4.3.2</version>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
        <dependency>
            <groupId>joda-time</groupId>
            <artifactId>joda-time</artifactId>
        </dependency>
```

```
    </dependencies>

</project>
```

1.6.2、配置文件

application.properties

```
spring.application.name = itcast-rocketmq
server.port = 18082

# rocketmq相关配置
spring.rocketmq.nameServer=192.168.31.81:9876
spring.rocketmq.producer.group=tanhua

# Redis相关配置
spring.redis.jedis.pool.max-wait = 5000ms
spring.redis.jedis.pool.max-Idle = 100
spring.redis.jedis.pool.min-Idle = 10
spring.redis.timeout = 10s
spring.redis.cluster.nodes =
192.168.31.81:6379,192.168.31.81:6380,192.168.31.81:6381
spring.redis.cluster.max-redirects=5

# mongodb相关配置
spring.data.mongodb.uri=mongodb://192.168.31.81:27017/tanhua
```

log4j.properties

```
log4j.rootLogger=DEBUG,A1

log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=[%t] [%c]-[%p] %m%n
```

1.6.3、RecommendQuanZi

存储到MongoDB的中的实体结构。

```
package com.tanhua.recommend.pojo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.bson.types.ObjectId;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class RecommendQuanZi {

    private ObjectId id;
    private Long userId;// 用户id
    private Long publishId; //动态id，需要转化为Long类型
    private Double score; //得分
    private Long date; //时间戳
```

```
}

```

1.6.4、QuanZiMsgConsumer

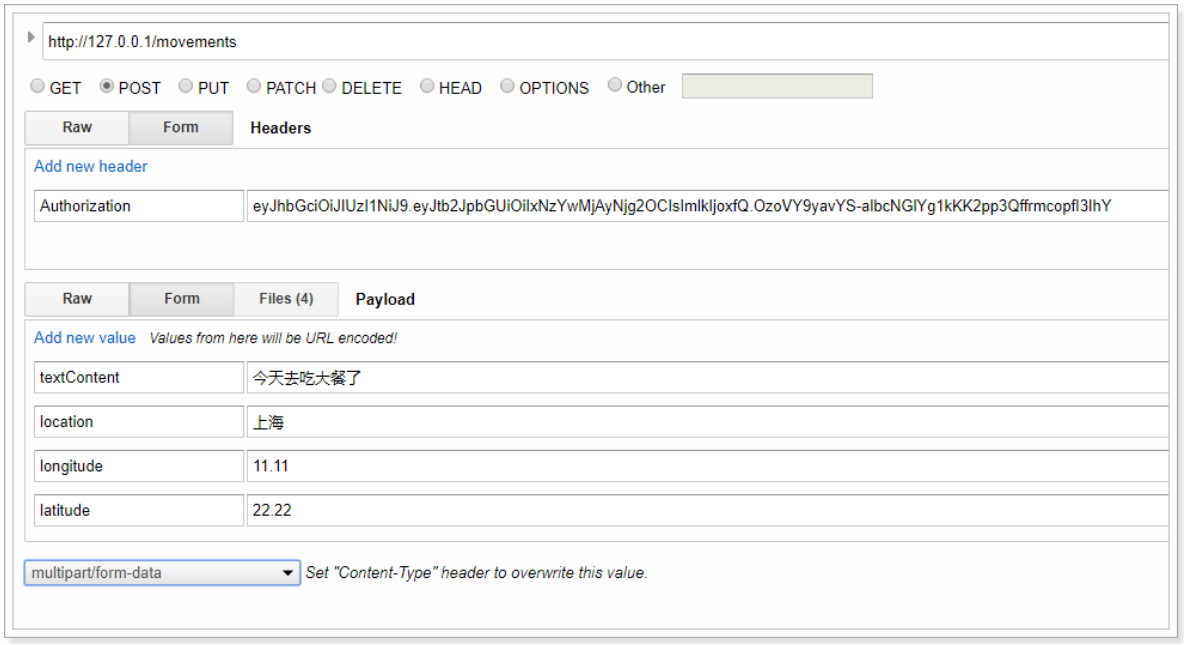
```
package com.tanhua.recommend.msg;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.tanhua.recommend.pojo.Publish;
import com.tanhua.recommend.pojo.RecommendQuanZi;
import org.apache.commons.lang3.StringUtils;
import org.apache.rocketmq.spring.annotation.RocketMQMessageListener;
import org.apache.rocketmq.spring.core.RocketMQListener;
import org.bson.types.ObjectId;
import org.joda.time.DateTime;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.stereotype.Component;
import org.springframework.util.CollectionUtils;

@Component
@RocketMQMessageListener(topic = "tanhua-quanzi",
        consumerGroup = "tanhua-quanzi-consumer")
public class QuanZiMsgConsumer implements RocketMQListener<String> {

    private static final ObjectMapper MAPPER = new ObjectMapper();

    private static final Logger LOGGER =
LoggerFactory.getLogger(QuanZiMsgConsumer.class);

    @Autowired
    private MongoTemplate mongoTemplate;

    @Override
    public void onMessage(String msg) {
        try {
            JsonNode jsonNode = MAPPER.readTree(msg);

            Long userId = jsonNode.get("userId").asLong();
            Long pid = jsonNode.get("pid").asLong();
            String publishId = jsonNode.get("publishId").asText();
            Integer type = jsonNode.get("type").asInt();

            //1-发动态，2-浏览动态， 3-点赞， 4-喜欢， 5-评论，6-取消点赞，7-取消喜
欢
            RecommendQuanZi recommendQuanZi = new RecommendQuanZi();
            recommendQuanZi.setUserId(userId);
            recommendQuanZi.setId(ObjectId.get());
            recommendQuanZi.setDate(System.currentTimeMillis());
            recommendQuanZi.setPublishId(pid);

            switch (type) {
                case 1: {
                    int score = 0;
```

```
                    Publish publish = this.mongoTemplate.findById(new ObjectId(publishId), Publish.class);
                    if (StringUtils.length(publish.getText()) < 50) {
                        score += 1;
                    } else if (StringUtils.length(publish.getText()) < 100) {
                        score += 2;
                    } else if (StringUtils.length(publish.getText()) >= 100) {
                        score += 3;
                    }

                    if (!CollectionUtils.isEmpty(publish.getMedias())) {
                        score += publish.getMedias().size();
                    }

                    recommendQuanZi.setScore(Double.valueOf(score));

                    break;
                }
                case 2: {
                    recommendQuanZi.setScore(1d);
                    break;
                }
                case 3: {
                    recommendQuanZi.setScore(5d);
                    break;
                }
                case 4: {
                    recommendQuanZi.setScore(8d);
                    break;
                }
                case 5: {
                    recommendQuanZi.setScore(10d);
                    break;
                }
                case 6: {
                    recommendQuanZi.setScore(-5d);
                    break;
                }
                case 7: {
                    recommendQuanZi.setScore(-8d);
                    break;
                }
                default: {
                    recommendQuanZi.setScore(0d);
                    break;
                }
            }

            String collectionName = "recommend_quanzi_" + new DateTime().toString("yyyyMMdd");
            this.mongoTemplate.save(recommendQuanZi, collectionName);

        } catch (Exception e) {
            LOGGER.error("处理消息失败~" + msg, e);
        }
    }
```

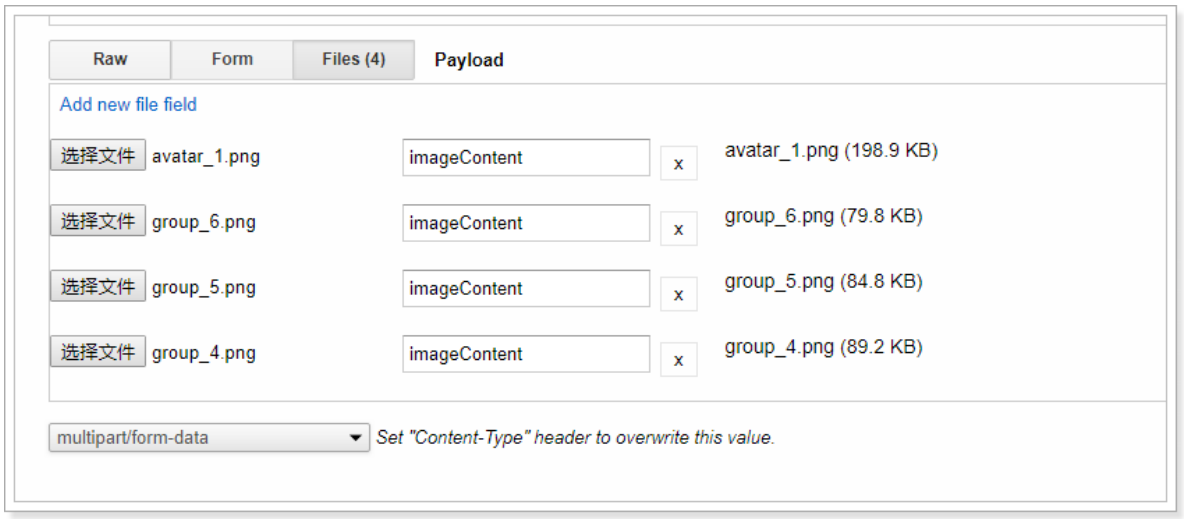
```
105 }
106
```

1.7、测试

1.7.1、发布动态



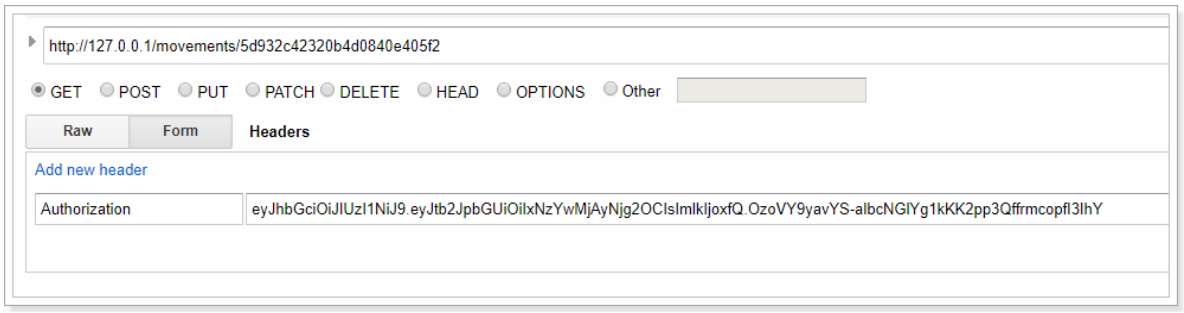
发布4张图片：



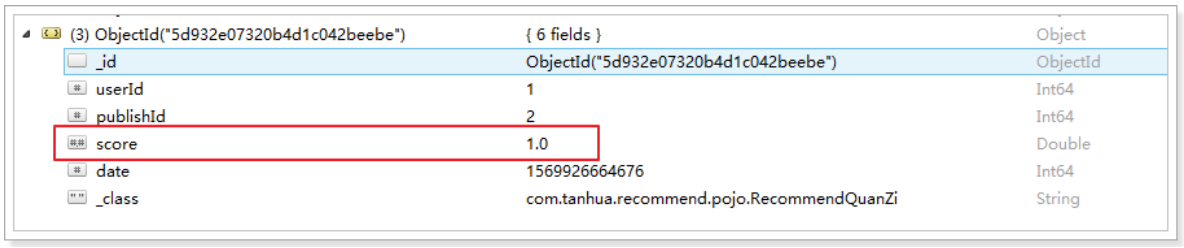
数据：

消息处理：

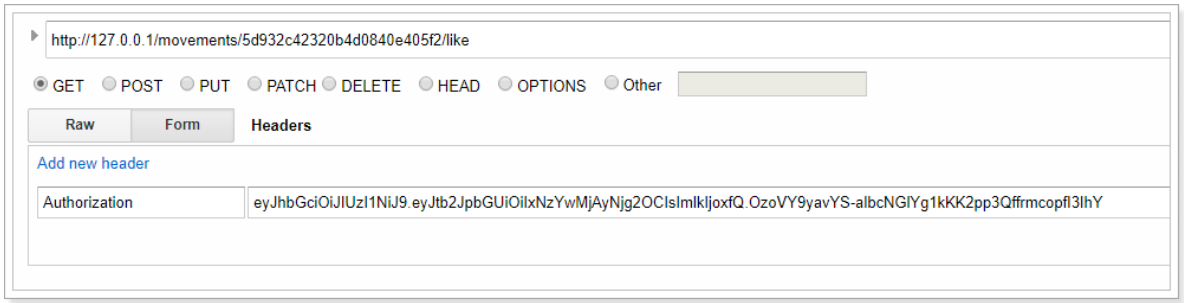
1.7.2、浏览动态



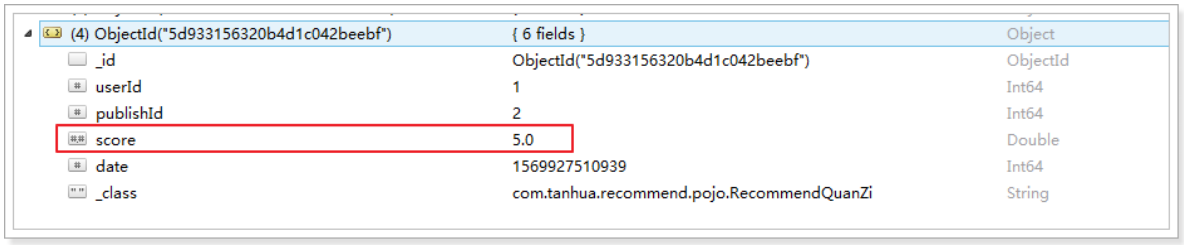
消息处理：



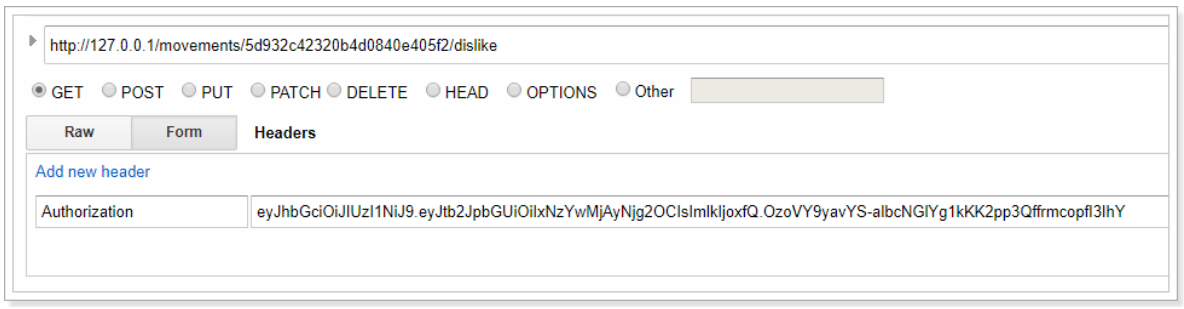
1.7.3、点赞



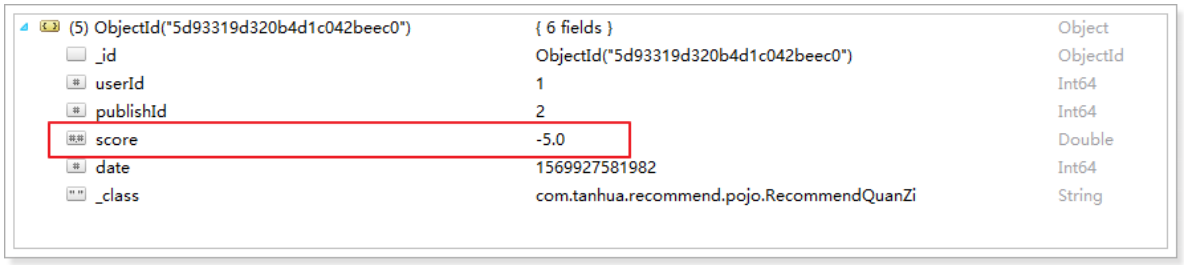
消息处理：



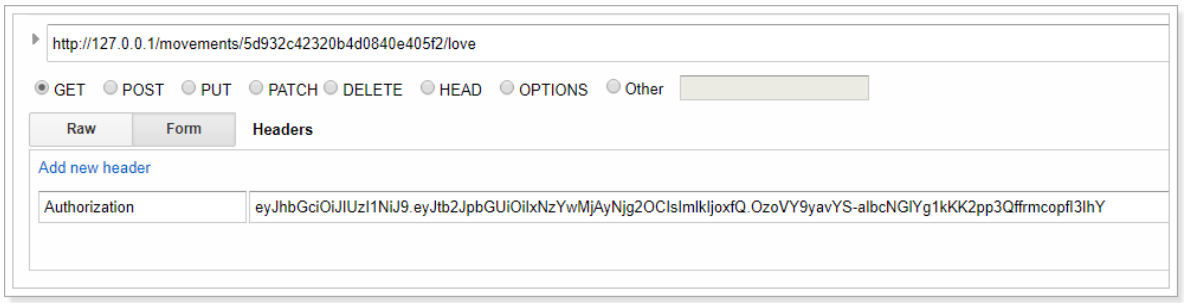
1.7.4、取消点赞



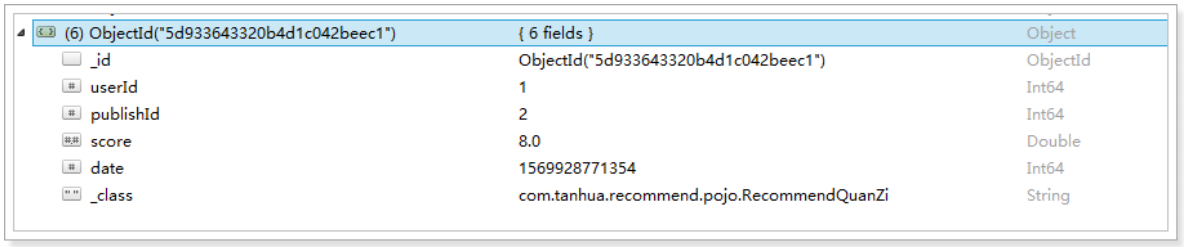
消息处理：



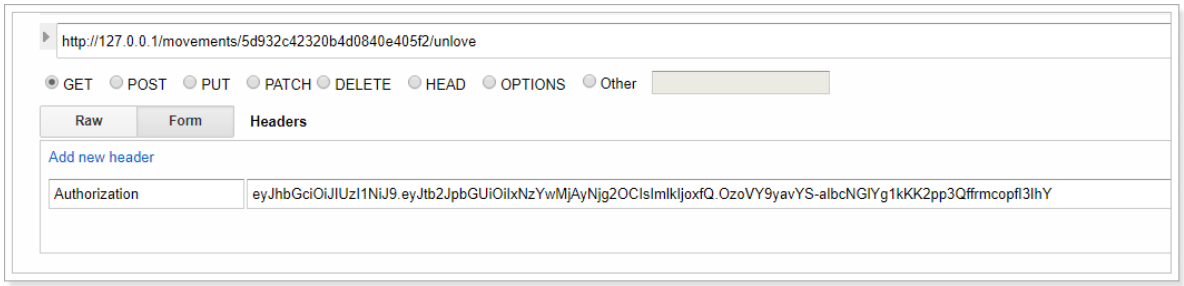
1.7.5、喜欢



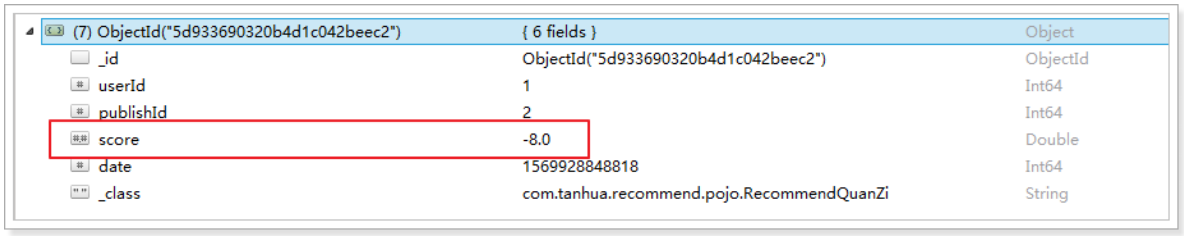
消息处理：



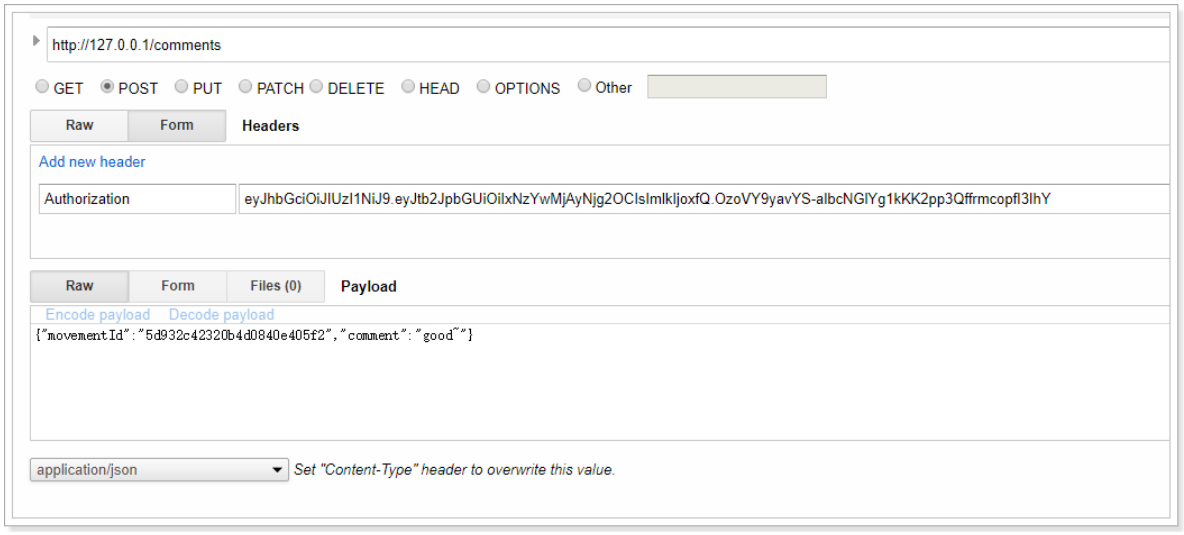
1.7.6、取消喜欢



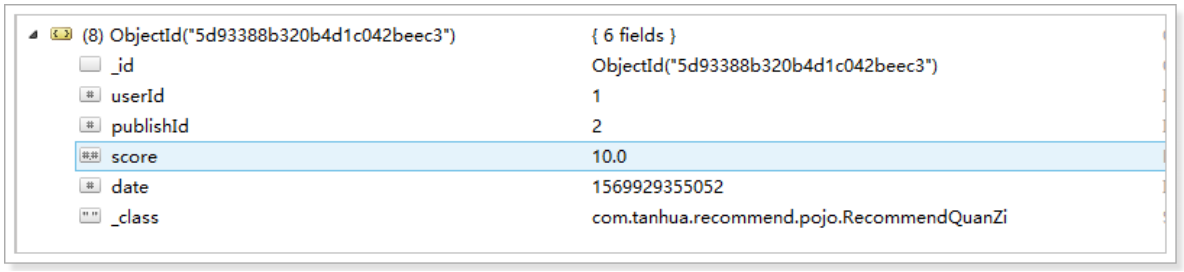
消息处理：



1.7.7、评论



消息处理：



2、推荐系统

构造数据：

```
    @Test
    public void testSavePublish() {
        for (int i = 0; i < 100; i++) {
            Publish publish = new Publish();
            publish.setUserId(RandomUtils.nextLong(1,10));
            publish.setPid(Long.valueOf(1000 + i));
            publish.setLocationName("上海市");
            publish.setSeeType(1);
            publish.setText("今天天气不错~ " + i);
            publish.setMedias(Arrays.asList("http://itcast-tanhua.oss-cn-shanghai.aliyuncs.com/images/2019/10/01/15699262101875720.png",
                    "http://itcast-tanhua.oss-cn-shanghai.aliyuncs.com/images/2019/10/01/15699262107836768.png",
                    "http://itcast-tanhua.oss-cn-shanghai.aliyuncs.com/images/2019/10/01/15699262108584571.png",
                    "http://itcast-tanhua.oss-cn-shanghai.aliyuncs.com/images/2019/10/01/15699262109221190.png"));
            String publishId = this.quanZiApi.savePublish(publish);
            System.out.println(publishId);
        }
    }

    @Test
    public void testSaveRecommend(){
        for (int i = 0; i < 1000; i++) {
            RecommendQuanZi recommendQuanZi = new RecommendQuanZi();
```

```
            recommendQuanZi.setDate(System.currentTimeMillis());
            recommendQuanZi.setId(ObjectId.get());

 recommendQuanZi.setScore(Double.valueOf(RandomUtils.nextInt(0,15)));
            recommendQuanZi.setUserId(RandomUtils.nextLong(1,10));
            recommendQuanZi.setPublishId(RandomUtils.nextLong(1000,1099));

            this.mongoTemplate.save(recommendQuanZi,
"recommend_quanzi_20191001");
        }
    }
```

2.1、搭建系统

搭建系统itcast-tanhua-spark用于实现推荐功能。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>itcast-tanhua</artifactId>
        <groupId>cn.itcast.tanhua</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <packaging>jar</packaging>

    <artifactId>itcast-tanhua-spark</artifactId>

    <dependencies>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-mllib_2.12</artifactId>
            <version>2.4.3</version>
            <!--<scope>runtime</scope>-->

            <exclusions>
                <exclusion>
                    <groupId>javax.servlet</groupId>
                    <artifactId>javax.servlet-api</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.mongodb.spark</groupId>
            <artifactId>mongo-spark-connector_2.12</artifactId>
            <version>2.4.1</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
            <!--<scope>provided</scope>-->
        </dependency>
        <dependency>
```

```
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
        <dependency>
            <groupId>redis.clients</groupId>
            <artifactId>jedis</artifactId>
            <version>3.1.0</version>
        </dependency>
    </dependencies>

</project>
```

2.2、配置文件app.properties

```
spark.mongodb.input.uri =
mongodb://192.168.31.81:27017/tanhua.recommend_quanzi_20191001?
readPreference=primaryPreferred

redis.cluster.nodes =
192.168.31.81:6379,192.168.31.81:6380,192.168.31.81:6381
```

2.3、MLlibRecommend

使用Spark MLlib进行推荐。

```
package com.tanhua.spark.mongo;

import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.mllib.recommendation.ALS;
import org.apache.spark.mllib.recommendation.MatrixFactorizationModel;
import org.apache.spark.mllib.recommendation.Rating;
import scala.Tuple2;

public class MLlibRecommend {

    public MatrixFactorizationModel bestModel(JavaPairRDD<Long, Rating>
ratings){
        //统计有用户数量和动态数量以及用户对动态的评分数目
        Long numRatings = ratings.count();
        Long numUsers = ratings.map(v1 ->
(v1._2()).user()).distinct().count();
        Long numMovies = ratings.map(v1 ->
(v1._2()).product()).distinct().count();
        System.out.println("用户：" + numUsers + "动态：" + numMovies + "评
论：" + numRatings);

        //将样本评分表以key值切分成3个部分，分别用于训练 (60%，并加入用户评分)， 校验
(20%)， and 测试 (20%)
        //该数据在计算过程中要多次应用到，所以cache到内存

        Integer numPartitions = 4; // 分区数
        // 训练集
        JavaRDD<Rating> training = ratings
                .filter(v -> v._1() < 6)
                .values()
```

```
                .repartition(numPartitions)
                .cache();

        // 校验集
        JavaRDD<Rating> validation = ratings
                .filter(v -> v._1() >= 6 && v._1() < 8)
                .values()
                .repartition(numPartitions).cache();

        // 测试集
        JavaRDD<Rating> test = ratings
                .filter(v -> v._1() >= 8)
                .values()
                .cache();

        Long numTraining = training.count();
        Long numValidation = validation.count();
        Long numTest = test.count();
        System.out.println("训练集: " + numTraining + " 校验集: " +
numValidation + " 测试集: " + numTest);

        //训练不同参数下的模型，并在校验集中验证，获取最佳参数下的模
        int[] ranks = new int[]{10, 11, 12};
//        double[] lambdas = new double[]{0.01, 0.03, 0.1, 0.3, 1, 3};
        double[] lambdas = new double[]{0.01};
//        int[] numIters = new int[]{8, 9, 10, 11, 12, 13, 14, 15};
        int[] numIters = new int[]{8, 9, 10};

        MatrixFactorizationModel bestModel = null;
        double bestValidationRmse = Double.MAX_VALUE;
        int bestRank = 0;
        double bestLambda = -0.01;
        int bestNumIter = 0;

        for (int rank : ranks) {
            for (int numIter : numIters) {
                for (double lambda : lambdas) {
                    MatrixFactorizationModel model =
ALS.train(training.rdd(), rank, numIter, lambda);
                    Double validationRmse = computeRmse(model, validation,
numValidation);
                    System.out.println("RMSE(校验集) = " + validationRmse +
", rank = " + rank + ", lambda = " + lambda + ", numIter = " + numIter);

                    if (validationRmse < bestValidationRmse) {
                        bestModel = model;
                        bestValidationRmse = validationRmse;
                        bestRank = rank;
                        bestLambda = lambda;
                        bestNumIter = numIter;
                    }
                }
            }
        }



        double testRmse = computeRmse(bestModel, test, numTest);
```

```
        System.out.println("测试数据集在 最佳训练模型 rank = " + bestRank + ", lambda = " + bestLambda + ", numIter = " + bestNumIter + ", RMSE = " + testRmse);

        // 计算均值
        Double meanRating = training.union(validation).mapToDouble(v -> v.rating()).mean();

        // 计算标准误差值
        Double baselineRmse = Math.sqrt(test.map(v -> (meanRating - v.rating()) * (meanRating - v.rating())).reduce((v1, v2) -> (v1 + v2) / numTest));

        // 计算准确率提升了多少
        double improvement = (baselineRmse - testRmse) / baselineRmse * 100;

        System.out.println("最佳训练模型的准确率提升了：" + String.format("%.2f", improvement) + "%.");


        // 构建最佳训练模型
        bestModel = ALS.train(ratings.values().rdd(), bestRank, bestNumIter, bestLambda);

        return bestModel;
    }

    /**
     * 校验集预测数据和实际数据之间的均方根误差
     **/
    public  Double computeRmse(MatrixFactorizationModel model, JavaRDD<Rating> data, Long n) {
        // 进行预测
        JavaRDD<Rating> predictions = model.predict(data.mapToPair(v -> new Tuple2<>(v.user(), v.product())));

        JavaRDD<Tuple2<Double, Double>> predictionsAndRatings = predictions
                .mapToPair(v -> new Tuple2<>(new Tuple2<>(v.user(), v.product()), v.rating()))
                .join(data.mapToPair(v -> new Tuple2<>(new Tuple2<>(v.user(), v.product()), v.rating()))).values();

        Double reduce = predictionsAndRatings.map(v -> (v._1 - v._2) * (v._1 - v._2))
                .reduce((v1, v2) -> (v1 + v2) / n);
        //正平方根
        return Math.sqrt(reduce);
    }
}

```

2.4、SparkMongoDB

实现读取MongoDB，并且使用MLlib的推荐模型进行推荐，最终将推荐结果数据写入到Redis中。

```java
package com.tanhua.spark.mongo;

import com.mongodb.spark.MongoSpark;
import com.mongodb.spark.rdd.api.java.JavaMongoRDD;
import org.apache.commons.lang3.StringUtils;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.mllib.recommendation.MatrixFactorizationModel;
import org.apache.spark.mllib.recommendation.Rating;
import org.bson.Document;
import redis.clients.jedis.HostAndPort;
import redis.clients.jedis.JedisCluster;
import scala.Tuple2;

import java.io.InputStream;
import java.util.*;

public class SparkMongoDB {

    public static void main(String[] args) throws Exception{

        //读取外部的配置文件
        InputStream inputStream =
SparkMongoDB.class.getClassLoader().getResourceAsStream("app.properties");
        Properties properties = new Properties();
        properties.load(inputStream);

        //构建Spark配置
        SparkConf sparkConf = new SparkConf()
                .setAppName("SparkMongoDB")
                .setMaster("local[*]")
                .set("spark.mongodb.input.uri",
properties.getProperty("spark.mongodb.input.uri"));

        //构建Spark上下文
        JavaSparkContext jsc = new JavaSparkContext(sparkConf);

        //加载MongoDB中的数据
        JavaMongoRDD<Document> rdd = MongoSpark.load(jsc);

//        rdd.foreach(document -> System.out.println(document.toJson()));

        //在数据中有同一个用户对不同的动态进行评价，需要进行合并操作
        JavaRDD<Document> values = rdd.mapToPair(document -> {
            Integer user = document.getLong("userId").intValue();
            Integer product = document.getLong("publishId").intValue();
            return new Tuple2<>(user + "_" + product, document);
        }).reduceByKey((v1, v2) -> {
            Double score = v1.getDouble("score") + v2.getDouble("score");
            v1.put("score", score);
            return v1;
        }).values();

        //得到数据中的用户id集合
        List<Long> userIdList = rdd.map(v1 ->
v1.getLong("userId")).distinct().collect();
```

```
56
57  //        values.foreach(document ->
    System.out.println(document.toJson()));
58
59          //按照日期对10进行取模作为key，Rating对象作为value，获取到数据用于后续的数据
    处理
60          JavaPairRDD<Long, Rating> ratings = values.mapToPair(document -> {
61              Integer user = document.getLong("userId").intValue();
62              Integer product = document.getLong("publishId").intValue();
63              Double score = document.getDouble("score");
64              Long date = document.getLong("date");
65              Rating rating = new Rating(user, product, score);
66              return new Tuple2<>(date % 10, rating);
67          });
68
69          //通过MLlib模型进行推荐，获取到最优的推荐模型
70          MLlibRecommend mLlibRecommend = new MLlibRecommend();
71          MatrixFactorizationModel bestModel =
    mLlibRecommend.bestModel(ratings);
72
73          //构建Redis环境
74          String redisClusterNodes =
    properties.getProperty("redis.cluster.nodes");
75          String[] redisNodes = redisClusterNodes.split(",");
76          Set<HostAndPort> nodes = new HashSet<>();
77          for (String redisNode : redisNodes) {
78              String[] hostAndPorts = redisNode.split(":");
79              nodes.add(new HostAndPort(hostAndPorts[0],
    Integer.valueOf(hostAndPorts[1])));
80          }
81          JedisCluster jedisCluster = new JedisCluster(nodes);
82
83          //分别对每一个用户进行推荐，推荐20个动态信息
84          for (Long userId : userIdList) {
85              Rating[] recommendProducts =
    bestModel.recommendProducts(userId.intValue(), 20);
86
87              List<Integer> products = new ArrayList<>();
88              for (Rating rating : recommendProducts) {
89                  products.add(rating.product());
90              }
91
92              //存储到redis
93              String key = "QUANZI_PUBLISH_RECOMMEND_" + userId;
94              System.out.println(key);
95              jedisCluster.set(key, StringUtils.join(products, ','));
96          }
97
98          //关闭连接
99          jedisCluster.close();
100
101     }
102 }
103
```

2.5、测试

```
192.168.31.81:6380> get QUANZI_PUBLISH_RECOMMEND_2
"1003,1012,1062,1079,1077,1081,1053,1097,1018,1085,1048,1026,1020,1042,1049,1005,1094,1022,1074,1035"
192.168.31.81:6380> get QUANZI_PUBLISH_RECOMMEND_1
-> Redirected to slot [4156] located at 192.168.31.81:6379
"1004,1050,1035,1003,1027,1016,1083,1041,1092,1037,1012,1030,1059,1061,1006,1009,1038,1043,1013,1090"
192.168.31.81:6379> get QUANZI_PUBLISH_RECOMMEND_3
-> Redirected to slot [12414] located at 192.168.31.81:6381
"1004,1030,1089,1043,1073,1002,1085,1022,1016,1023,1071,1098,1058,1037,1069,1034,1012,1083,1077,1060"
192.168.31.81:6381>
```

可以看到已经有推荐的数据写入到了Redis中。

3、修改查询逻辑

之前是通过MongoDB直接查询，而现在需要先从Redis进行命中，如果未命中则需要进行MongoDB查询。

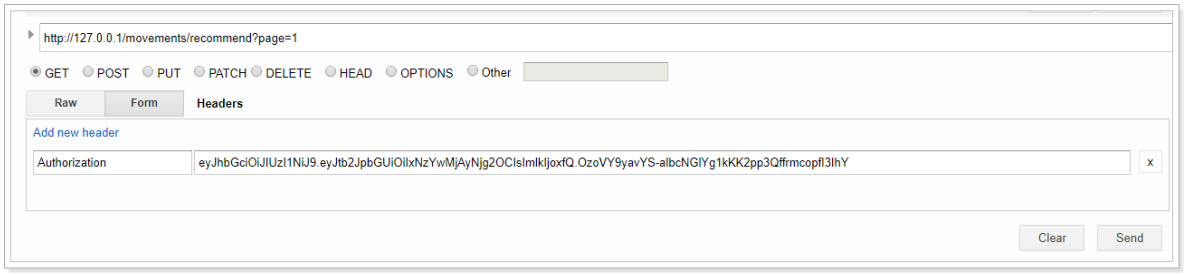
修改server工程中的MovementsService类型：

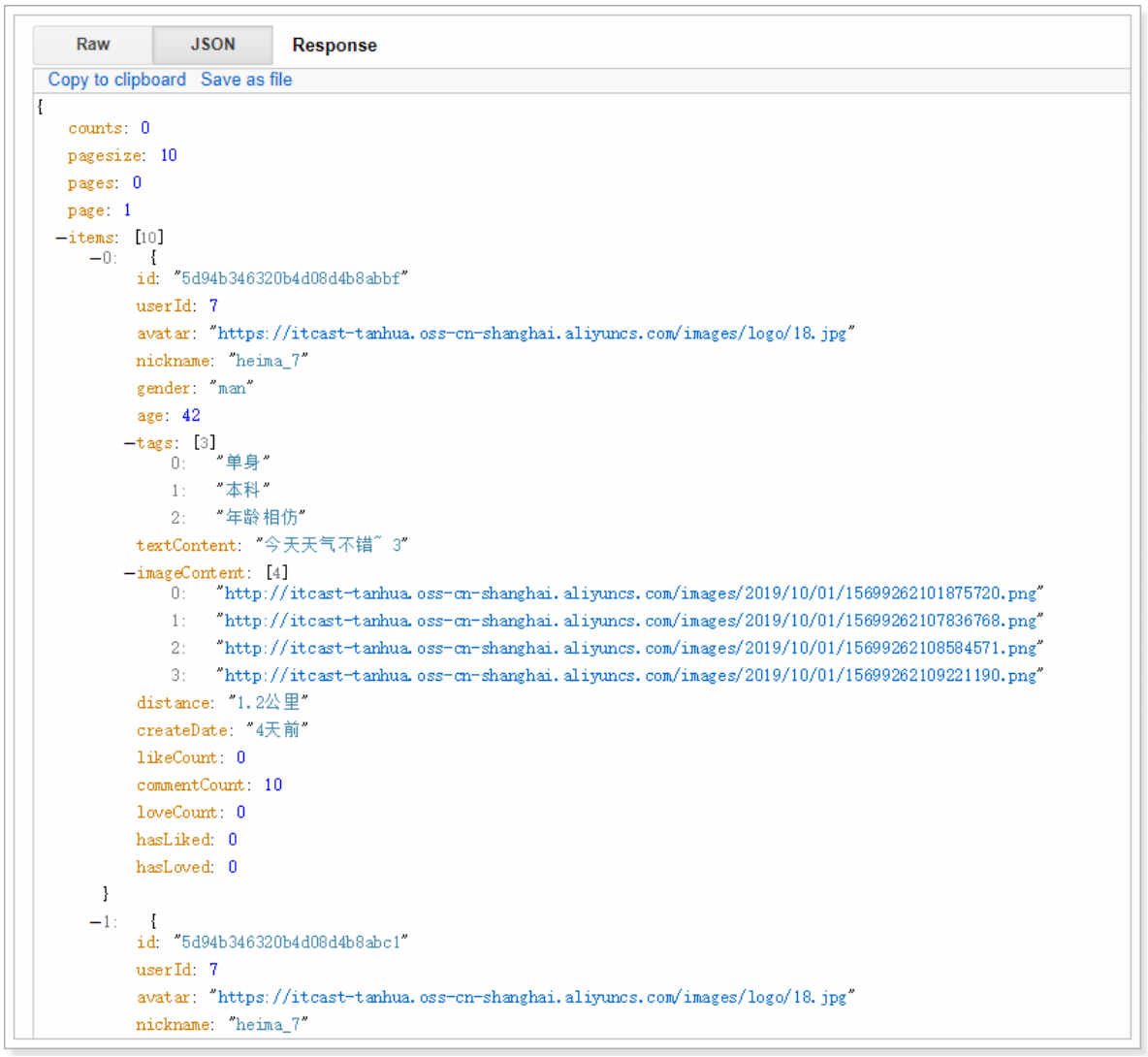
```
 /**
     * 查询动态
     *
     * @param page
     * @param pageSize
     * @return
     */
    public PageResult queryPublishList(Integer page, Integer pageSize, boolean isRecommend) {
        PageResult pageResult = new PageResult();
        //获取当前的登录信息
        User user = UserThreadLocal.get();


        PageInfo<Publish> pageInfo = null;
        if (isRecommend) { //推荐动态逻辑处理
            // 查询Redis
            String value = this.redisTemplate.opsForValue().get("QUANZI_PUBLISH_RECOMMEND_" + user.getId());
            if (StringUtils.isNotEmpty(value)) {
                String[] pids = StringUtils.split(value, ',');
                int startIndex = (page - 1) * pageSize;
                if(startIndex < pids.length){
                    int endIndex = startIndex + pageSize - 1;
                    if (endIndex >= pids.length) {
                        endIndex = pids.length - 1;
                    }

                    List<Long> pidList = new ArrayList<>();
                    for (int i = startIndex; i <= endIndex; i++) {
                        pidList.add(Long.valueOf(pids[i]));
                    }

                    List<Publish> publishList = this.quanZiApi.queryPublishByPids(pidList);
                    pageInfo = new PageInfo<>();
                    pageInfo.setRecords(publishList);
                }
            }
        }
```

```
38
39          if (null == pageInfo) {
40              Long userId = isRecommend ? null : user.getId();
41              pageInfo = this.quanZiApi.queryPublishList(userId, page,
    pageSize);
42          }
43
44          pageResult.setPagesize(pageSize);
45          pageResult.setPage(page);
46          pageResult.setCounts(0);
47          pageResult.setPages(0);
48
49          List<Publish> records = pageInfo.getRecords();
50
51          if (CollectionUtils.isEmpty(records)) {
52              //没有动态信息
53              return pageResult;
54          }
55
56          List<Movements> movementsList = new ArrayList<>();
57          for (Publish record : records) {
58              Movements movements = new Movements();
59
60              movements.setId(record.getId().toHexString());
61              movements.setImageContent(record.getMedias().toArray(new
    String[]{}));
62              movements.setTextContent(record.getText());
63              movements.setUserId(record.getUserId());
64              movements.setCreateDate(RelativeDateFormat.format(new
    Date(record.getCreated())));
65
66              movementsList.add(movements);
67          }
68
69          List<Long> userIds = new ArrayList<>();
70          for (Movements movements : movementsList) {
71              if (!userIds.contains(movements.getUserId())) {
72                  userIds.add(movements.getUserId());
73              }
74
75          }
76
77          QueryWrapper<UserInfo> queryWrapper = new QueryWrapper<>();
78          queryWrapper.in("user_id", userIds);
79          List<UserInfo> userInfos =
    this.userInfoService.queryList(queryWrapper);
80          for (Movements movements : movementsList) {
81              for (UserInfo userInfo : userInfos) {
82                  if (movements.getUserId().longValue() ==
    userInfo.getUserId().longValue()) {
83                      this.fillValueToMovements(movements, userInfo);
84                      break;
85                  }
86              }
87          }
88
89          pageResult.setItems(movementsList);
90          return pageResult;
```

```
91          }
```

测试：





可以看到，已经查询到了动态数据。

4、小视频推荐

小视频的推荐和动态推荐的实现逻辑非常的类似。

4.1、增加自增id

```
1  package com.tanhua.dubbo.server.pojo;
2
3  import lombok.AllArgsConstructor;
4  import lombok.Data;
5  import lombok.NoArgsConstructor;
6  import org.bson.types.ObjectId;
7  import org.springframework.data.mongodb.core.mapping.Document;
```

```
8
9  import java.util.List;
10
11 @Data
12 @NoArgsConstructor
13 @AllArgsConstructor
14 @Document(collection = "video")
15 public class Video implements java.io.Serializable {
16
17     private static final long serialVersionUID = -3136732836884933873L;
18
19     private ObjectId id; //主键id
20     private Long vid;
21     private Long userId;
22     private String text; //文字
23     private String picUrl; //视频封面文件
24     private String videoUrl; //视频文件
25     private Long created; //创建时间
26     private Integer seeType; // 谁可以看，1-公开，2-私密，3-部分可见，4-不给谁看
27     private List<Long> seeList; //部分可见的列表
28     private List<Long> notSeeList; //不给谁看的列表
29     private String longitude; //经度
30     private String latitude; //纬度
31     private String locationName; //位置名称
32 }
33
```

修改VideoApiImpl逻辑：

```
1      @Override
2      public Boolean saveVideo(Video video) {
3          if (video.getUserId() == null) {
4              return false;
5          }
6
7          video.setId(ObjectId.get());
8          video.setCreated(System.currentTimeMillis());
9
10         //生成vid
11         video.setVid(this.idService.createId("video",
   video.getId().toHexString()));
12
13         this.mongoTemplate.save(video);
14         return true;
15     }
```

4.2、动态计分规则

- 发布+2
- 点赞 +5
- 评论 + 10

4.3、发送消息

4.3.1、VideoMQService

```
```

```java
package com.tanhua.server.service;

import com.alibaba.dubbo.config.annotation.Reference;
import com.tanhua.dubbo.server.api.QuanZiApi;
import com.tanhua.dubbo.server.api.VideoApi;
import com.tanhua.dubbo.server.pojo.Publish;
import com.tanhua.dubbo.server.pojo.Video;
import com.tanhua.server.pojo.User;
import com.tanhua.server.utils.UserThreadLocal;
import org.apache.rocketmq.spring.core.RocketMQTemplate;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.HashMap;
import java.util.Map;

@Service
public class VideoMQService {

    private static final Logger LOGGER = LoggerFactory.getLogger(VideoMQService.class);

    @Autowired
    private RocketMQTemplate rocketMQTemplate;

    @Reference(version = "1.0.0")
    private VideoApi videoApi;

    /**
     * 发布小视频消息
     *
     * @return
     */
    public Boolean videoMsg(String videoId) {
        return this.sendMsg(videoId, 1);
    }

    /**
     * 点赞小视频
     *
     * @return
     */
    public Boolean likeVideoMsg(String videoId) {
        return this.sendMsg(videoId, 2);
    }

    /**
     * 取消点赞小视频
     *
     * @return
     */
    public Boolean disLikeVideoMsg(String videoId) {
        return this.sendMsg(videoId, 3);
    }

    /**
```

```
     * 评论小视频
     *
     * @return
     */
    public Boolean commentVideoMsg(String videoId) {
        return this.sendMsg(videoId, 4);
    }

    /**
     * 发送小视频操作相关的消息
     *
     * @param videoId
     * @param type     1-发动态，2-点赞，  3-取消点赞，4-评论
     * @return
     */
    private Boolean sendMsg(String videoId, Integer type) {
        try {
            User user = UserThreadLocal.get();

            Video video = this.videoApi.queryVideoById(videoId);

            //构建消息
            Map<String, Object> msg = new HashMap<>();
            msg.put("userId", user.getId());
            msg.put("date", System.currentTimeMillis());
            msg.put("videoId", videoId);
            msg.put("vid", video.getVid());
            msg.put("type", type);

            this.rocketMQTemplate.convertAndSend("tanhua-video", msg);
        } catch (Exception e) {
            LOGGER.error("发送消息失败! videoId = " + videoId + ", type = " +
type, e);
            return false;
        }

        return true;
    }
}

```

4.3.2、VideoController

```
package com.tanhua.server.controller;

import com.tanhua.server.service.MovementsService;
import com.tanhua.server.service.VideoMQService;
import com.tanhua.server.service.VideoService;
import com.tanhua.server.vo.PageResult;
import org.apache.commons.lang3.StringUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import java.util.Map;
```

```java

@RestController
@RequestMapping("smallVideos")
public class VideoController {

    @Autowired
    private VideoService videoService;

    @Autowired
    private MovementsService movementsService;

    @Autowired
    private CommentsController commentsController;

    @Autowired
    private VideoMQService videoMQService;

    /**
     * 发布小视频
     *
     * @param picFile
     * @param videoFile
     * @return
     */
    @PostMapping
    public ResponseEntity<Void> saveVideo(@RequestParam(value = "videoThumbnail", required = false) MultipartFile picFile,
                                          @RequestParam(value = "videoFile", required = false) MultipartFile videoFile) {
        try {
            String id = this.videoService.saveVideo(picFile, videoFile);
            if (StringUtils.isNotEmpty(id)) {
                //发送消息
                this.videoMQService.videoMsg(id);

                return ResponseEntity.ok(null);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }

    /**
     * 查询小视频列表
     *
     * @param page
     * @param pageSize
     * @return
     */
    @GetMapping
    public ResponseEntity<PageResult> queryVideoList(@RequestParam(value = "page", defaultValue = "1") Integer page,
                                                      @RequestParam(value = "pagesize", defaultValue = "10") Integer pageSize) {
        try {
            if (page <= 0) {
```

```
                page = 1;
            }
            PageResult pageResult = this.videoService.queryVideoList(page, pageSize);
            if (null != pageResult) {
                return ResponseEntity.ok(pageResult);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }

    /**
     * 视频点赞
     *
     * @param videoId 视频id
     * @return
     */
    @PostMapping("/{id}/like")
    public ResponseEntity<Long> likeComment(@PathVariable("id") String videoId) {
        try {
            Long likeCount = this.movementsService.likeComment(videoId);
            if (likeCount != null) {
                this.videoMQService.likeVideoMsg(videoId);
                return ResponseEntity.ok(likeCount);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }

    /**
     * 取消点赞
     *
     * @param videoId
     * @return
     */
    @PostMapping("/{id}/dislike")
    public ResponseEntity<Long> disLikeComment(@PathVariable("id") String videoId) {
        try {
            Long likeCount = this.movementsService.cancelLikeComment(videoId);
            if (null != likeCount) {
                this.videoMQService.disLikeVideoMsg(videoId);
                return ResponseEntity.ok(likeCount);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }
```

```
119
120     /**
121      * 评论点赞
122      *
123      * @param publishId
124      * @return
125      */
126     @PostMapping("/comments/{id}/like")
127     public ResponseEntity<Long> commentsLikeComment(@PathVariable("id")
String publishId) {
128         try {
129             Long likeCount = this.movementsService.likeComment(publishId);
130             if (likeCount != null) {
131                 return ResponseEntity.ok(likeCount);
132             }
133         } catch (Exception e) {
134             e.printStackTrace();
135         }
136         return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
137     }
138
139     /**
140      * 评论取消点赞
141      *
142      * @param publishId
143      * @return
144      */
145     @PostMapping("/comments/{id}/dislike")
146     public ResponseEntity<Long> disCommentsLikeComment(@PathVariable("id")
String publishId) {
147         try {
148             Long likeCount =
this.movementsService.cancelLikeComment(publishId);
149             if (null != likeCount) {
150                 return ResponseEntity.ok(likeCount);
151             }
152         } catch (Exception e) {
153             e.printStackTrace();
154         }
155         return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
156     }
157
158     /**
159      * 提交评论
160      *
161      * @param param
162      * @param videoId
163      * @return
164      */
165     @PostMapping("/{id}/comments")
166     public ResponseEntity<Void> saveComments(@RequestBody Map<String,
String> param,
167                                              @PathVariable("id") String
videoId) {
168         param.put("movementId", videoId);
```

```
        ResponseEntity<Void> entity = this.commentsController.saveComments(param);

        if (entity.getStatusCode().is2xxSuccessful()) {
            //发送消息
            this.videoMQService.commentVideoMsg(videoId);
        }

        return entity;
    }

    /**
     * 评论列表
     */
    @GetMapping("/{id}/comments")
    public ResponseEntity<PageResult> queryCommentsList(@PathVariable("id") String videoId,
 @RequestParam(value = "page", defaultValue = "1") Integer page,
 @RequestParam(value = "pagesize", defaultValue = "10") Integer pagesize) {
        return this.commentsController.queryCommentsList(videoId, page, pagesize);
    }

    /**
     * 视频用户关注
     */
    @PostMapping("/{id}/userFocus")
    public ResponseEntity<Void> saveUserFocusComments(@PathVariable("id") Long userId) {
        try {
            Boolean bool = this.videoService.followUser(userId);
            if (bool) {
                return ResponseEntity.ok(null);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }

    /**
     * 视频用户关注
     */
    @PostMapping("/{id}/userUnFocus")
    public ResponseEntity<Void> saveUserUnFocusComments(@PathVariable("id") Long userId) {
        try {
            Boolean bool = this.videoService.disFollowUser(userId);
            if (bool) {
                return ResponseEntity.ok(null);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }
}

```

4.4、接收消息

4.4.1、RecommendVideo

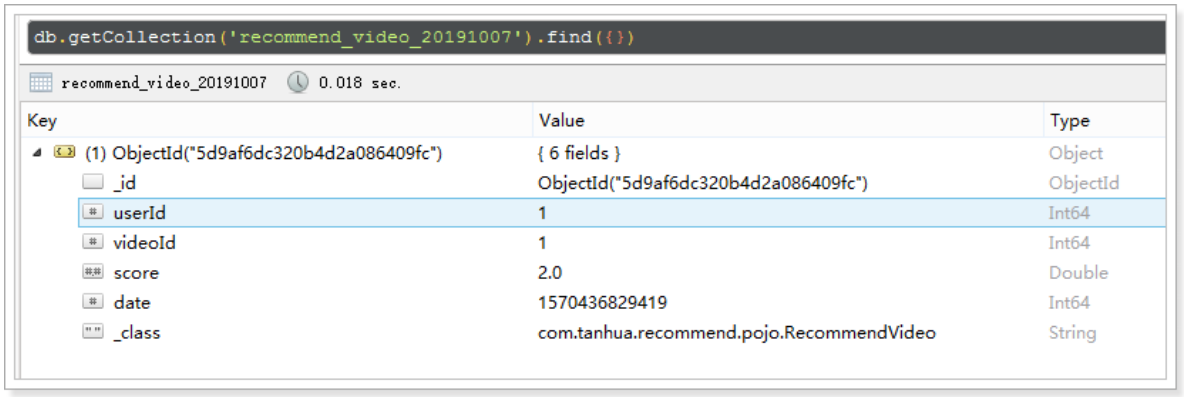
```
package com.tanhua.recommend.pojo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.bson.types.ObjectId;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class RecommendVideo {

    private ObjectId id;
    private Long userId;// 用户id
    private Long videoId; //视频id，需要转化为Long类型
    private Double score; //得分
    private Long date; //时间戳
}

```

4.4.2、VideoMsgConsumer

```
package com.tanhua.recommend.msg;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.tanhua.recommend.pojo.Publish;
import com.tanhua.recommend.pojo.RecommendQuanZi;
import com.tanhua.recommend.pojo.RecommendVideo;
import org.apache.commons.lang3.StringUtils;
import org.apache.rocketmq.spring.annotation.RocketMQMessageListener;
import org.apache.rocketmq.spring.core.RocketMQListener;
import org.bson.types.ObjectId;
import org.joda.time.DateTime;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.stereotype.Component;
import org.springframework.util.CollectionUtils;

@Component
@RocketMQMessageListener(topic = "tanhua-video",
        consumerGroup = "tanhua-video-consumer")
public class VideoMsgConsumer implements RocketMQListener<String> {

```

```
    private static final ObjectMapper MAPPER = new ObjectMapper();

    private static final Logger LOGGER =
LoggerFactory.getLogger(VideoMsgConsumer.class);

    @Autowired
    private MongoTemplate mongoTemplate;

    @Override
    public void onMessage(String msg) {
        try {
            JsonNode jsonNode = MAPPER.readTree(msg);

            Long userId = jsonNode.get("userId").asLong();
            Long vid = jsonNode.get("vid").asLong();
            String videoId = jsonNode.get("videoId").asText();
            Integer type = jsonNode.get("type").asInt();

            //1-发动态，2-点赞，  3-取消点赞，4-评论
            RecommendVideo recommendVideo = new RecommendVideo();
            recommendVideo.setUserId(userId);
            recommendVideo.setId(ObjectId.get());
            recommendVideo.setDate(System.currentTimeMillis());
            recommendVideo.setVideoId(vid);

            switch (type) {
                case 1: {
                    recommendVideo.setScore(2d);
                    break;
                }
                case 2: {
                    recommendVideo.setScore(5d);
                    break;
                }
                case 3: {
                    recommendVideo.setScore(-5d);
                    break;
                }
                case 4: {
                    recommendVideo.setScore(10d);
                    break;
                }
                default: {
                    recommendVideo.setScore(0d);
                    break;
                }
            }

            String collectionName = "recommend_video_" + new
DateTime().toString("yyyyMMdd");
            this.mongoTemplate.save(recommendVideo, collectionName);

        } catch (Exception e) {
            LOGGER.error("处理小视频消息失败~" + msg, e);
        }
    }
}

```

4.4.3、测试



4.5、构造数据

```
package com.tanhua.dubbo.server.api;

import com.tanhua.dubbo.server.pojo.Video;
import org.apache.commons.lang3.RandomUtils;
import org.bson.types.ObjectId;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest
public class TestVideoApi {

    @Autowired
    private VideoApi videoApi;

    @Autowired
    private MongoTemplate mongoTemplate;

    @Test
    public void testSaveVideo() {
        for (int i = 0; i < 100; i++) {
            Video video = new Video();
            video.setId(ObjectId.get());
            video.setUserId(RandomUtils.nextLong(1,10));
            video.setVid(Long.valueOf(1000 + i));
            video.setLocationName("上海市");
            video.setSeeType(1);
            video.setPicUrl("http://itcast-tanhua.oss-cn-
shanghai.aliyuncs.com/images/2019/10/07/15704367549907417.png");

 video.setVideoUrl("http://192.168.31.81:8888/group1/M00/00/02/wKgfUV2a9pmA
QgIlACYJuhDRF3g362.mp4");
            video.setCreated(System.currentTimeMillis());
            String videoId = this.videoApi.saveVideo(video);
            System.out.println(videoId);
        }
```

```
    }

    @Test
    public void testSaveRecommend(){
        for (int i = 0; i < 1000; i++) {
            RecommendVideo recommendQuanZi = new RecommendVideo();
            recommendQuanZi.setDate(System.currentTimeMillis());
            recommendQuanZi.setId(ObjectId.get());
 recommendQuanZi.setScore(Double.valueOf(RandomUtils.nextInt(0,15)));
            recommendQuanZi.setUserId(RandomUtils.nextLong(1,10));
            recommendQuanZi.setVideoId(RandomUtils.nextLong(1000,1099));

            this.mongoTemplate.save(recommendQuanZi,
"recommend_video_20191007");
        }
    }

}

```

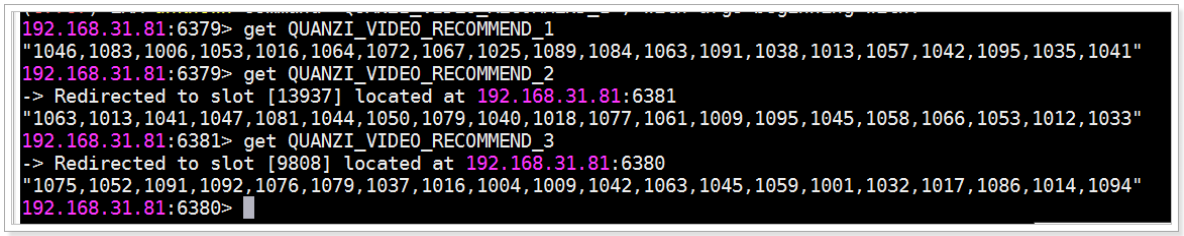
4.6、实现推荐

```
package com.tanhua.spark.mongo;

import com.mongodb.spark.MongoSpark;
import com.mongodb.spark.rdd.api.java.JavaMongoRDD;
import org.apache.commons.lang3.StringUtils;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.mllib.recommendation.MatrixFactorizationModel;
import org.apache.spark.mllib.recommendation.Rating;
import org.bson.Document;
import redis.clients.jedis.HostAndPort;
import redis.clients.jedis.JedisCluster;
import scala.Tuple2;

import java.io.InputStream;
import java.util.*;

public class SparkVideoMongoDB {

    public static void main(String[] args) throws Exception{

        //读取外部的配置文件
        InputStream inputStream =
SparkVideoMongoDB.class.getClassLoader().getResourceAsStream("app.properti
es");
        Properties properties = new Properties();
        properties.load(inputStream);

        //构建Spark配置
        SparkConf sparkConf = new SparkConf()
                .setAppName("SparkVideoMongoDB")
                .setMaster("local[*]")
```

```
                .set("spark.mongodb.input.uri",
properties.getProperty("spark.video.mongodb.input.uri"));

        //构建Spark上下文
        JavaSparkContext jsc = new JavaSparkContext(sparkConf);

        //加载MongoDB中的数据
        JavaMongoRDD<Document> rdd = MongoSpark.load(jsc);

//        rdd.foreach(document -> System.out.println(document.toJson()));

        //在数据中有同一个用户对不同的小视频进行评价，需要进行合并操作
        JavaRDD<Document> values = rdd.mapToPair(document -> {
            Integer user = document.getLong("userId").intValue();
            Integer product = document.getLong("videoId").intValue();
            return new Tuple2<>(user + "_" + product, document);
        }).reduceByKey((v1, v2) -> {
            Double score = v1.getDouble("score") + v2.getDouble("score");
            v1.put("score", score);
            return v1;
        }).values();

        //得到数据中的用户id集合
        List<Long> userIdList = rdd.map(v1 ->
v1.getLong("userId")).distinct().collect();

//        values.foreach(document ->
System.out.println(document.toJson()));

        //按照日期对10进行取模作为key，Rating对象作为value，获取到数据用于后续的数据
处理
        JavaPairRDD<Long, Rating> ratings = values.mapToPair(document -> {
            Integer user = document.getLong("userId").intValue();
            Integer product = document.getLong("videoId").intValue();
            Double score = document.getDouble("score");
            Long date = document.getLong("date");
            Rating rating = new Rating(user, product, score);
            return new Tuple2<>(date % 10, rating);
        });

        //通过MLlib模型进行推荐，获取到最优的推荐模型
        MLlibRecommend mLlibRecommend = new MLlibRecommend();
        MatrixFactorizationModel bestModel =
mLlibRecommend.bestModel(ratings);

        //构建Redis环境
        String redisClusterNodes =
properties.getProperty("redis.cluster.nodes");
        String[] redisNodes = redisClusterNodes.split(",");
        Set<HostAndPort> nodes = new HashSet<>();
        for (String redisNode : redisNodes) {
            String[] hostAndPorts = redisNode.split(":");
            nodes.add(new HostAndPort(hostAndPorts[0],
Integer.valueOf(hostAndPorts[1])));
        }
        JedisCluster jedisCluster = new JedisCluster(nodes);

        //分别对每一个用户进行推荐，推荐20个小视频信息
```

```
        for (Long userId : userIdList) {
            Rating[] recommendProducts =
bestModel.recommendProducts(userId.intValue(), 20);

            List<Integer> products = new ArrayList<>();
            for (Rating rating : recommendProducts) {
                products.add(rating.product());
            }

            //存储到redis
            String key = "QUANZI_VIDEO_RECOMMEND_" + userId;
            jedisCluster.set(key, StringUtils.join(products, ','));
        }

        //关闭连接
        jedisCluster.close();
        jedisCluster.close();

    }
}

```

测试：



4.7、修改查询逻辑

修改VideoService的实现：

```
public PageResult queryVideoList(Integer page, Integer pageSize) {

        User user = UserThreadLocal.get();

        PageResult pageResult = new PageResult();
        pageResult.setPage(page);
        pageResult.setPagesize(pageSize);
        pageResult.setPages(0);
        pageResult.setCounts(0);

        PageInfo<Video> pageInfo = null;

        //先从Redis进行命中，如果命中则返回推荐列表，如果未命中查询默认列表
        String redisValue =
this.redisTemplate.opsForValue().get("QUANZI_VIDEO_RECOMMEND_" +
user.getId());
        if (StringUtils.isNotEmpty(redisValue)) {
            String[] pids = StringUtils.split(redisValue, ',');
            int startIndex = (page - 1) * pageSize;
            if (startIndex < pids.length) {
                int endIndex = startIndex + pageSize - 1;
                if (endIndex >= pids.length) {
```

```
                    endIndex = pids.length - 1;
                }

                List<Long> vidList = new ArrayList<>();
                for (int i = startIndex; i <= endIndex; i++) {
                    vidList.add(Long.valueOf(pids[i]));
                }

                List<Video> videoList = this.videoApi.queryVideoListByPids(vidList);
                pageInfo = new PageInfo<>();
                pageInfo.setRecords(videoList);
            }
        }

        if(null == pageInfo){
            pageInfo = this.videoApi.queryVideoList(page, pageSize);
        }

        List<Video> records = pageInfo.getRecords();
        List<VideoVo> videoVoList = new ArrayList<>();
        List<Long> userIds = new ArrayList<>();
        for (Video record : records) {
            VideoVo videoVo = new VideoVo();

            videoVo.setUserId(record.getUserId());
            videoVo.setCover(record.getPicUrl());
            videoVo.setVideoUrl(record.getVideoUrl());
            videoVo.setId(record.getId().toHexString());
            videoVo.setSignature("我就是我~");

            Long commentCount = this.quanZiApi.queryCommentCount(videoVo.getId(), 2);
            videoVo.setCommentCount(commentCount == null ? 0 : commentCount.intValue()); // 评论数

            String followUserKey = "VIDEO_FOLLOW_USER_" + user.getId() + "_" + videoVo.getUserId();
            videoVo.setHasFocus(this.redisTemplate.hasKey(followUserKey) ? 1 : 0); //是否关注

            String userKey = "QUANZI_COMMENT_LIKE_USER_" + user.getId() + "_" + videoVo.getId();
            videoVo.setHasLiked(this.redisTemplate.hasKey(userKey) ? 1 : 0); //是否点赞（1是，0否）

            String key = "QUANZI_COMMENT_LIKE_" + videoVo.getId();
            String value = this.redisTemplate.opsForValue().get(key);
            if (StringUtils.isNotEmpty(value)) {
                videoVo.setLikeCount(Integer.valueOf(value)); //点赞数
            } else {
                videoVo.setLikeCount(0);
            }

            if (!userIds.contains(record.getUserId())) {
                userIds.add(record.getUserId());
            }

```

```
            videoVoList.add(videoVo);
        }

        QueryWrapper<UserInfo> queryWrapper = new QueryWrapper();
        queryWrapper.in("user_id", userIds);
        List<UserInfo> userInfos =
this.userInfoService.queryList(queryWrapper);
        for (VideoVo videoVo : videoVoList) {
            for (UserInfo userInfo : userInfos) {
                if (videoVo.getUserId().longValue() ==
userInfo.getUserId().longValue()) {

                    videoVo.setNickname(userInfo.getNickName());
                    videoVo.setAvatar(userInfo.getLogo());

                    break;
                }
            }

        }

        pageResult.setItems(videoVoList);
        return pageResult;
    }
```