kafka实战应用&文章自动审核

今日目标

熟悉kafka的封装技巧

熟悉阿里审核图片和文本内容审核

完成自媒体文章审核代码

完成自媒体端发布文章发送消息

完成admin端接收消息并自动审核

1 kafka封装

1.1 功能需求

消息对于现代软件项目来说，占有很重要的地位；同时市场上也发展处ActiveMq、RabbitMQ、Kafka、RocketMQ、Pulsar等众多优秀的框架；这些优秀的框架都由自身的特点和擅长的业务领域，在大数据领域中Kafka目前是使用较多的框架，Pulsar是一个后起之秀，目前处于一个快速发展的状态，有望能够成为下一代中间件的黑马。在本案例中我们选择使用Kafka作为内部消息通知的框架，以适应项目中大数据量的高吞吐、实时流计算等功能实现。

1.2 定义

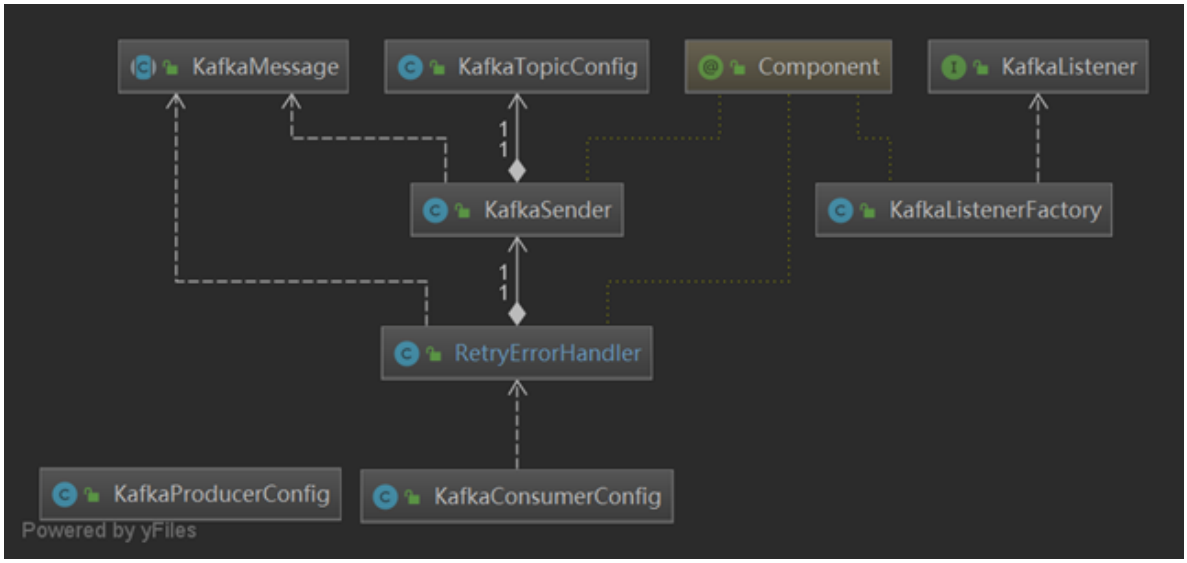
1.2.1 约束定义

（1）Topic命名约束

Topic分为单类和混合类消息，不同类的消息命名约束如下：

- 单类：heima.topic.[自定义名称].sigle
- 混合类：heima.topic.[自定义名称].bus

1.3 实现设计



- KafkaProducerConfig自动配置Kafka消费者
- KafkaConsumerConfig自动配置Kafka消费者
- RetryErrorHandler实现消费者处理消息失败后重新发送到消息队列
- KafkaMessage实现对发送的消息包装，提供重试次数、分类等信息
- KafkaSender实现消息的统一发送入口功能
- KafkaTopicConfig自动装载topic名称信息
- KafkaListener提供自动注册消息消费监听接口类
- KafkaListenerFactory提供启动时自动注册实现了KafkaListener的消息消费者

1.4 开发实现

1.4.1 配置文件

Kafka功能有独立的配置文件，放置在src\main\resources\kafka.properties，相关的值在maven_*.properties中配置。

```
#kafka config
kafka.hosts=localhost:9092
kafka.group=heima.${profiles.name}.${spring.application.name}

# 单消息通道，需要以sigle结尾
kafka.topic.admin-test=${kafka.topic.admin-test}
```

1.4.2 KafkaMessage

创建类com.heima.common.kafka.KafkaMessage。KafkaMessage是一个抽象类包含记录当前消息重发处理的次数retry、消息类型type、第一次创建消息的时间time信息。

```
/**
 * Kafka消息
 */
public abstract class KafkaMessage<T> {

    // 尝试次数
    @Getter
    int retry;
    // 生成时间
    @Getter
    long time = System.currentTimeMillis();
```

```
    // 消息类型
    String type;
    // 消息实体数据
    @Setter
    @Getter
    T data;
    public KafkaMessage(){}
    public KafkaMessage(T data){
        this.data = data;
    }

    public void addRetry(){
        this.retry++;
    }
    // 获取消息类型
    protected abstract String getType();
}
```

1.4.3 KafkaListener

创建类com.heima.common.kafka.KafkaListener。KafkaListener是一个接口，继承ConsumerAwareMessageListener（提供Consumer信息和自动提交offsets功能）接口。

- topic方法用于返回监听器监听的topic名称
- factory方法用于指定监听器容器的创建工厂
- group方法用于指定监听器的groupid，目前没用

```
/**
 * 消息监听实现接口
 */
public interface KafkaListener<K,V> extends ConsumerAwareMessageListener<K,V> {

    String topic();

    default String factory(){
        return "defaultKafkaListenerContainerFactory";
    }

    default  String group(){ return "default";}

}
```

1.4.4 KafkaTopicConfig

创建类：com.heima.common.kafka.KafkaTopicConfig。KafkaTopicConfig用于自动装入kafka.properties文件中的kafka.topic.*信息

```
@Data
@Configuration
@ConfigurationProperties(prefix="kafka.topic")
@PropertySource("classpath:kafka.properties")
public class KafkaTopicConfig {
    String userLogin;
    String userLogout;
    String userRefresh;
    String userRegister;
    String hotArticle;
}
```

1.4.5 KafkaProducerConfig

创建类com.heima.common.kafka.KafkaProducerConfig。KafkaProducerConfig类是自动化配置类，定义了默认的Producer工厂，以及KafkaTemplate，并约束了消息的类型为String,大小不超过16M。

```
@Data
@Configuration
@EnableKafka
@ConfigurationProperties(prefix="kafka")
@PropertySource("classpath:kafka.properties")
public class KafkaProducerConfig {
    private static final int MAX_MESSAGE_SIZE = 16* 1024 * 1024;
    private String hosts;

    @Autowired(required = false)
    private ProducerListener<String, String> producerListener;

    @Bean
    public DefaultKafkaProducerFactory<String, String> producerFactory() {
        Map<String, Object> props = new HashMap<>();
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, this.getHosts());
        props.put(ProducerConfig.RETRIES_CONFIG, 10);
        props.put(ProducerConfig.RETRY_BACKOFF_MS_CONFIG, 5_000);
        props.put(ProducerConfig.BUFFER_MEMORY_CONFIG,3*MAX_MESSAGE_SIZE);
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);
        props.put(ProducerConfig.MAX_REQUEST_SIZE_CONFIG,3*MAX_MESSAGE_SIZE);
        props.put(ProducerConfig.COMPRESSION_TYPE_CONFIG, "lz4");
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);
        props.put(ProducerConfig.BATCH_SIZE_CONFIG, 256 * 1024);
        return new DefaultKafkaProducerFactory<>( props);
    }

    @Bean
    public KafkaTemplate<String, String> kafkaTemplate(ProducerFactory
producerFactory) {
        KafkaTemplate<String, String> t = new KafkaTemplate<>(producerFactory);
        if (this.producerListener != null) {
            t.setProducerListener(this.producerListener);
        }
        return t;
    }
```

```
}
```

1.4.6 KafkaSender

创建类com.heima.common.kafka.KafkaSender。KafkaSender类是所有发送消息的方法统一管理器，其实现通过kafkaTemplate发送。

```
@Component
public class KafkaSender {

    Logger logger = LoggerFactory.getLogger(KafkaSender.class);

    @Autowired
    KafkaTemplate<String, String> kafkaTemplate;
    @Autowired
    ObjectMapper mapper;
    @Autowired
    KafkaTopicConfig kafkaTopicConfig;

    /**
     * 发送一个消息
     * @param topic
     * @param key
     * @param message
     */
    public void sendMesssage(String topic,String key,KafkaMessage<?> message){
        try {
            this.kafkaTemplate.send(topic, key,
mapper.writeValueAsString(message));
        }catch (Exception e){
            logger.error("send message to [{}] error:",topic,e);
        }
    }

    /**
     * 发送一个不包装的消息
     * 只能是内部使用，拒绝业务上使用
     * @param topic
     * @param key
     * @param message
     */
    public void sendMesssageNoWrap(String topic,String key,String message){
        try {
            this.kafkaTemplate.send(topic, key, message);
        }catch (Exception e){
            logger.error("send message to [{}] error:",topic,e);
        }
    }
}
```

1.4.7 RetryErrorHandler

创建类com.heima.common.kafka.RetryErrorHandler。RetryErrorHandler类用于在消费者解析消息出现错误时，重新放回消息到队列中，并设置超过一个小时或者超过10次处理错误的消息丢弃，避免消息无限滚动；然后这类消息可以通过日志搜索查找出数据补偿重试。

```
@Component
public class RetryErrorHandler extends LoggingErrorHandler {
    private static Logger logger =
LoggerFactory.getLogger(RetryErrorHandler.class);
    private static  final  int RETRY_COUNT = 10;
    private static  final  int TIME_OUT = 3_600_000;//1个小时超时

    @Autowired
    KafkaSender sender;
    @Autowired
    ObjectMapper mapper;

    @Override
    public void handle(Exception thrownException, ConsumerRecord<?, ?> record) {
        super.handle(thrownException, record);
        if (record != null) {
            try{
                KafkaMessage<?> message =
mapper.readValue((String)record.value(),KafkaMessage.class);
                message.addRetry();
                long time = System.currentTimeMillis()-message.getTime();
                if(message.getRetry()>RETRY_COUNT||time>TIME_OUT){
                    logger.info("超时或者尝试{}次后，抛弃消息[topic:{}]
[{}]",RETRY_COUNT,record.topic(),record.value());
                }else{
                    this.sender.sendMesssage(record.topic(),
(String)record.key(),message);
                    logger.info("处理失败重新回滚到队列[retry:{}][topic:{}][key:
{}]",message.getRetry(),record.topic(),record.key());
                }
            }catch (Exception e){
                sender.sendMesssageNoWrap(record.topic(),(String) record.key(),
(String) record.value());
            }

        }
    }

}
```

1.4.8 KafkaProducerConfig

创建类com.heima.common.kafka.KafkaProducerConfig。KafkaProducerConfig主要配置消费者监听器，配置重试器、错误处理器等信息，同时设置group消费者。

```
@Data
@Configuration
@EnableKafka
@ConfigurationProperties(prefix="kafka")
@PropertySource("classpath:kafka.properties")
public class KafkaConsumerConfig {
    private static final int CONCURRENCY = 8;
    public final static Logger LOGGER =
LoggerFactory.getLogger(KafkaConsumerConfig.class);

    String hosts;
    String group;
```

```
    @Bean("defaultKafkaListenerContainerFactory")
    public ConcurrentKafkaListenerContainerFactory<String, String>
kafkaListenerContainerFactory(RetryErrorHandler retryErrorHandler) {
        ConcurrentKafkaListenerContainerFactory<String, String> factory = new
ConcurrentKafkaListenerContainerFactory<>();
        factory.setRetryTemplate(this.buildRetryTemplate());
        factory.setErrorHandler(retryErrorHandler);
        factory.getContainerProperties().setAckOnError(false);
        factory.setConsumerFactory(new DefaultKafkaConsumerFactory<>
(buildComsumerConfig()));
        factory.setConcurrency(KafkaConsumerConfig.CONCURRENCY);
        return factory;
    }

    protected Map<String, Object> buildComsumerConfig() {
        Map<String, Object> propsMap = new HashMap<>();
        propsMap.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, this.getHosts());
        propsMap.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, false);
        propsMap.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class);

 propsMap.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,StringDeserializer.
class);
        propsMap.put(ConsumerConfig.GROUP_ID_CONFIG, this.group);
        propsMap.put(ConsumerConfig.MAX_PARTITION_FETCH_BYTES_CONFIG,8 * 1024 *
1024);
        propsMap.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 90_000);
        return propsMap;
    }

    private RetryTemplate buildRetryTemplate() {
        RetryTemplate t = new RetryTemplate();
        ExponentialBackOffPolicy backOff = new ExponentialRandomBackOffPolicy();
        backOff.setInitialInterval(1000L);
        t.setBackOffPolicy(backOff);
        t.setRetryPolicy(new SimpleRetryPolicy(5));
        t.registerListener(new RetryListenerSupport() {
            @Override
            public <T, E extends Throwable> void onError(RetryContext context,
                                                         RetryCallback<T, E>
callback, Throwable throwable) {
                KafkaConsumerConfig.LOGGER.warn("Retry processing Kafka message
"
                        + context.getRetryCount() + " times", throwable);
            }
        });
        return t;
    }

}
```

1.4.9 KafkaListenerFactory

创建类com.heima.common.kafka.KafkaListenerFactory。KafkaListenerFactory类实现在构造之后扫描实现了的KafkaListener接口的Bean，并自动注册成消费者监听器。

```
@Component
public class KafkaListenerFactory implements InitializingBean {

    Logger logger = LoggerFactory.getLogger(KafkaListenerFactory.class);

    @Autowired
    DefaultListableBeanFactory defaultListableBeanFactory;

    @Override
    public void afterPropertiesSet() {
        Map<String,KafkaListener> map =
defaultListableBeanFactory.getBeansOfType(KafkaListener.class);
        for (String key : map.keySet()) {
            KafkaListener k = map.get(key);
            AbstractKafkaListenerContainerFactory factory =
(AbstractKafkaListenerContainerFactory)defaultListableBeanFactory.getBean(k.fact
ory());
            AbstractMessageListenerContainer container =
factory.createContainer(k.topic());
            container.setupMessageListener(k);
            String beanName = k.getClass().getSimpleName()+"AutoListener" ;
            defaultListableBeanFactory.registerSingleton(beanName,container);
            logger.info("add auto listener [{}]",beanName);
        }
    }
}
```

1.4.10 MessagesRegister

```
/**
 * 扫描所有的kafkamessage类
 */
@Log4j2
@Component
public class MessagesRegister implements InitializingBean {

    Map<String,Class> messages = Maps.newConcurrentMap();

    @Override
    public void afterPropertiesSet() throws Exception {
        Reflections reflections = new Reflections("com.heima");
        Set<Class<? extends KafkaMessage>> ms =
reflections.getSubTypesOf(KafkaMessage.class);
        if(ms!=null){
            ms.forEach(cla->{
                try {
                    Constructor<?>[] cs = cla.getConstructors();
                    KafkaMessage mess = null;
                    if (cs != null && cs.length > 0) {
                        Class[] temp = cs[0].getParameterTypes();
                        Object[] parms = new Object[temp.length];
                        for (int i = 0; i < temp.length; i++) {
                            if(temp[i].isPrimitive()){
                                if(temp[i].getName().contains("boolean")){
                                    parms[i]=false;
                                }else {
```

```
                                    parms[i] = 0;
                                }
                            }else{
                                parms[i]=null;
                            }
                        }
                        mess = (KafkaMessage) cs[0].newInstance(parms);
                    } else {
                        mess = (KafkaMessage) cla.newInstance();
                    }
                    String type = mess.getType();
                    messages.put(type,cla);
                }catch (Exception e){

 System.out.println(cla+"====================:"+cla.getConstructors()
[0].getParameterCount());
                    e.printStackTrace();
                }
            });
        }
        log.info("================================================");
        log.info("scan kafka message resultt[{}]",messages);
        log.info("================================================");
    }

    /**
     * 通过消息的类型名称，查找对应的class定义
     * @param type
     * @return
     */
    public Class<? extends KafkaMessage> findClassByType(String type){
        return this.messages.get(type);
    }

}
```

1.5 消息生产者

```
@SpringBootTest
@RunWith(SpringRunner.class)
public class KafkaTest {

    @Autowired
    KafkaTemplate<String, String> kafkaTemplate;

    @Test
    public void test(){
        try {
            this.kafkaTemplate.send("topic.test", "123key","123value");
            System.out.println("================================");
            Thread.sleep(500000);// 休眠等待消费者接收消息
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }
}
```

1.6 消息消费者

```
@Component
public class TestKafkaListener implements KafkaListener<String,String> {
    @Override
    public String topic () {
        return "topic.test";
    }
    @Override
    public void onMessage (ConsumerRecord< String, String > data, Consumer< ?, ?
> consumer){
        System.out.println("============receive test message:" + data);
    }
}
```

2 文章审核自动化

2.1 阿里云服务介绍

内容检测API开发准备

您在使用内容检测API之前，需要先注册阿里云账号，添加Access Key并签约云盾内容安全。

操作步骤

1. 前往阿里云官网注册账号。如果已有注册账号，请跳过此步骤。
2. 打开云盾内容安全产品试用页面，单击**立即开通**，正式开通服务。
3. 在AccessKey管理页面管理您的AccessKeyID和AccessKeySecret。

2.1.1 文本垃圾内容检测

文本垃圾内容检测接口说明

抽取工具类：

安装sdk:

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-core</artifactId>
    <version>3.4.1</version>
</dependency>
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-green</artifactId>
    <version>3.4.1</version>
</dependency>
```

需要首先在aliyun.properties文件中配置

```
aliyun.access-key=xxx自己填写
aliyun.secret=xxx自己填写
```

在heima-leadnews-common工程中创建类com.heima.common.aliyun.AliyunTextScanRequest

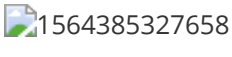
```
@Getter
@Setter
@Configuration
@ConfigurationProperties(prefix="aliyun")
@PropertySource("classpath:aliyun.properties")
public class AliyunTextScanRequest {

    private  String accessKey;
    private  String secret;

    public String textScanRequest(String content) throws Exception {
        IClientProfile profile = DefaultProfile.getProfile("cn-shanghai",
accessKey, secret);
        IAcsClient client = new DefaultAcsClient(profile);
        TextScanRequest textScanRequest = new TextScanRequest();
        textScanRequest.setAcceptFormat(FormatType.JSON); // 指定api返回格式
        textScanRequest.setHttpContentType(FormatType.JSON);
        textScanRequest.setMethod(com.aliyuncs.http.MethodType.POST); // 指定请求
方法
        textScanRequest.setEncoding("UTF-8");
        textScanRequest.setRegionId("cn-shanghai");
        List<Map<String, Object>> tasks = new ArrayList<Map<String, Object>>();
        Map<String, Object> task1 = new LinkedHashMap<String, Object>();
        task1.put("dataId", UUID.randomUUID().toString());
        /**
         * 待检测的文本，长度不超过10000个字符
         */
        task1.put("content", content);
        tasks.add(task1);
        JSONObject data = new JSONObject();

        /**
         * 检测场景，文本垃圾检测传递：antispam
         **/
        data.put("scenes", Arrays.asList("antispam"));
        data.put("tasks", tasks);
        System.out.println(JSON.toJSONString(data, true));
        textScanRequest.setHttpContent(data.toJSONString().getBytes("UTF-8"),
"UTF-8", FormatType.JSON);
        // 请务必设置超时时间
        textScanRequest.setConnectTimeout(3000);
        textScanRequest.setReadTimeout(6000);
        try {
            HttpResponse httpResponse = client.doAction(textScanRequest);
            if(httpResponse.isSuccess()){
                JSONObject scrResponse = JSON.parseObject(new
String(httpResponse.getHttpContent(), "UTF-8"));
                System.out.println(JSON.toJSONString(scrResponse, true));
                if (200 == scrResponse.getInteger("code")) {
                    JSONArray taskResults = scrResponse.getJSONArray("data");
                    for (Object taskResult : taskResults) {
```

```
                        if(200 == ((JSONObject)taskResult).getInteger("code")){
                            JSONArray sceneResults =
((JSONObject)taskResult).getJSONArray("results");
                            for (Object sceneResult : sceneResults) {
                                String scene =
((JSONObject)sceneResult).getString("scene");
                                String suggestion =
((JSONObject)sceneResult).getString("suggestion");
                                //根据scene和suggetion做相关处理
                                //suggestion == pass 未命中垃圾  suggestion ==
block 命中了垃圾，可以通过label字段查看命中的垃圾分类
                                System.out.println("args = [" + scene + "]");
                                System.out.println("args = [" + suggestion +
"]");
                                return suggestion;
                            }
                        }else{
                            System.out.println("task process fail:" +
((JSONObject)taskResult).getInteger("code"));
                        }
                    }
                } else {
                    System.out.println("detect not success. code:" +
scrResponse.getInteger("code"));
                }
            }else{
                System.out.println("response not success. status:" +
httpResponse.getStatus());
            }
        } catch (ClientException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

测试：

```
 @Autowired
private AliyunTextScanRequest aliyunTextScanRequest;

@Test
public void testTextScanRequest() throws Exception {
    String message = "阿里云，阿里巴巴集团旗下云计算品牌，全球卓越的云计算技术和服务提供商。
创立于2009年，在杭州、北京、硅谷等地设有研发中心和运营机构。";
    String response = aliyunTextScanRequest.textScanRequest(message);
    System.out.println(response);
}
```

通过

 1564385327658

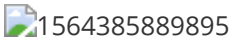
lebel:normal 正常文本

不通过：

当message内容为：包含了冰毒买卖

```
String message = "阿里云，阿里巴巴集团旗下冰毒买卖云计算品牌，全球卓越的云计算技术和服务提供商。创立于2009年，在杭州、北京、硅谷等地设有研发中心和运营机构。";
```

返回的结果为：

 1564385889895

- contraband：违禁

2.1.2 图片审核

创建工具类：com.heima.common.aliyun.AliyunImageScanRequest

```
@Getter
@Setter
@Configuration
@ConfigurationProperties(prefix="aliyun")
@PropertySource("classpath:aliyun.properties")
public class AliyunImageScanRequest {


    private  String accessKey;
    private  String secret;

    public String imageScanRequest(List<String> images) throws Exception {
        IClientProfile profile = DefaultProfile.getProfile("cn-
shanghai",accessKey,secret);
        DefaultProfile.addEndpoint("cn-shanghai", "cn-shanghai", "Green",
"green.cn-shanghai.aliyuncs.com");
        IAcsClient client = new DefaultAcsClient(profile);

        ImageSyncScanRequest imageSyncScanRequest = new ImageSyncScanRequest();
        // 指定api返回格式
        imageSyncScanRequest.setAcceptFormat(FormatType.JSON);
        // 指定请求方法
        imageSyncScanRequest.setMethod(MethodType.POST);
        imageSyncScanRequest.setEncoding("utf-8");
        //支持http和https
        imageSyncScanRequest.setProtocol(ProtocolType.HTTP);


        JSONObject httpBody = new JSONObject();
        /**
         * 设置要检测的场景，计费是按照该处传递的场景进行
         * 一次请求中可以同时检测多张图片，每张图片可以同时检测多个风险场景，计费按照场景计算
         * 例如：检测2张图片，场景传递porn,terrorism，计费会按照2张图片鉴黄，2张图片暴恐检
测计算
         * porn: porn表示色情场景检测
         */

        httpBody.put("scenes", Arrays.asList("logo","porn","ad","terrorism"));
```

```
        /**
         * 设置待检测图片， 一张图片一个task，
         * 多张图片同时检测时，处理的时间由最后一个处理完的图片决定。
         * 通常情况下批量检测的平均rt比单张检测的要长，一次批量提交的图片数越多，rt被拉长的概率越高
         * 这里以单张图片检测作为示例，如果是批量图片检测，请自行构建多个task
         */
        List<JSONObject> list = new ArrayList<JSONObject>();
        for (String image : images) {
            JSONObject task = new JSONObject();
            task.put("dataId", UUID.randomUUID().toString());
            //设置图片链接为上传后的url
            task.put("url", image);
            task.put("time", new Date());
            list.add(task);
        }

        httpBody.put("tasks", list);


 imageSyncScanRequest.setHttpContent(org.apache.commons.codec.binary.StringUtils.getBytesUtf8(httpBody.toJSONString()),
                "UTF-8", FormatType.JSON);

        /**
         * 请设置超时时间，服务端全链路处理超时时间为10秒，请做相应设置
         * 如果您设置的ReadTimeout 小于服务端处理的时间，程序中会获得一个read timeout 异常
         */
        imageSyncScanRequest.setConnectTimeout(3000);
        imageSyncScanRequest.setReadTimeout(10000);
        HttpResponse httpResponse = null;
        try {
            httpResponse = client.doAction(imageSyncScanRequest);
        } catch (Exception e) {
            e.printStackTrace();
        }

        //服务端接收到请求，并完成处理返回的结果
        if (httpResponse != null && httpResponse.isSuccess()) {
            JSONObject scrResponse = JSON.parseObject(org.apache.commons.codec.binary.StringUtils.newStringUtf8(httpResponse.getHttpContent()));
            System.out.println(JSON.toJSONString(scrResponse, true));
            int requestCode = scrResponse.getIntValue("code");
            //每一张图片的检测结果
            JSONArray taskResults = scrResponse.getJSONArray("data");
            if (200 == requestCode) {
                for (Object taskResult : taskResults) {
                    //单张图片的处理结果
                    int taskCode = ((JSONObject) taskResult).getIntValue("code");
                    //图片要检测的场景的处理结果，如果是多个场景，则会有每个场景的结果
                    JSONArray sceneResults = ((JSONObject) taskResult).getJSONArray("results");
                    if (200 == taskCode) {
```

```
                        for (Object sceneResult : sceneResults) {
                            String scene = ((JSONObject)
sceneResult).getString("scene");
                            String suggestion = ((JSONObject)
sceneResult).getString("suggestion");
                            //根据scene和suggetion做相关处理
                            //do something
                            System.out.println("scene = [" + scene + "]");
                            System.out.println("suggestion = [" + suggestion +
"]");
                            return suggestion;
                        }
                    } else {
                        //单张图片处理失败，原因是具体的情况详细分析
                        System.out.println("task process fail. task response:" +
JSON.toJSONString(taskResult));
                    }
                }
            } else {
                /**
                 * 表明请求整体处理失败，原因视具体的情况详细分析
                 */
                System.out.println("the whole image scan request failed.
response:" + JSON.toJSONString(scrResponse));
            }
        }
        return null;
    }
}
```

测试：

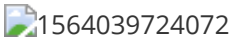
```
@Autowired
private AliyunImageScanRequest aliyunImageScanRequest;

@Test
public void testImageScanRequest(){
    try {
        List list = new ArrayList<>();

 list.add("http://47.94.7.85/group1/M00/00/00/rBENvl02ZtKAEgFqAACNdiGk7IM981.jpg
");
        String response = aliyunImageScanRequest.imageScanRequest(list);
        System.out.println(response);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

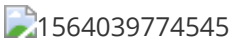
2.2 自媒体文章审核

2.2.1需求分析

当自媒体用户在自媒体端发表了一篇文章是需要通过审核通过以后才能在用户app端展示

2.2.2思路分析





流程图：参考资料文件夹中的：自媒体文章审核流程图.pdf

（1）自媒体端发表文章，发送消息到admin端，开始审核文章

（2）根据wmNews的文章id获取文章信息

（3）如果状态是人工审核，直接保存数据和创建索引

（4）审核通过后待发布的文章，判断发布时间是否大于当前时间，如果大于则直接保存数据

（5）审核状态为1（待审核）的文章

①根据文章标题和文章内容计算匹配度（人工复审，不匹配，匹配），不匹配需要发送通知消息

②调用阿里接口，审核文本

③调用阿里接口，审核图片

④审核通过以后查看发布的时间，如果发布时间大于当前时间则正常发布保存时间，否则修改状态为待发布

-审核通过以后需要保存数据，ap_article_config ap_article ap_article_content ap_author

-通知用户审核通过

-创建索引

2.2.3 代码实现

（1）mapper接口

修改com.heima.model.mappers.admin.AdChannelMapper，添加根据id查询频道方法

```
AdChannel selectByPrimaryKey(Integer id);
```

AdChannelMapper.xml

```
<select id="selectByPrimaryKey" resultMap="BaseResultMap"
parameterType="java.lang.Integer">
    select
    <include refid="Base_Column_List"/>
    from ad_channel
    where id = #{id}
</select>
```

修改com.heima.model.mappers.app.ApAuthorMapper，添加根据名称查询作者,添加方法

```
ApAuthor selectByAuthorName(String authorName);
void insert(ApAuthor apAuthor);
```

ApAuthorMapper.xml

```
<select id="selectByAuthorName" parameterType="java.lang.String"
resultMap="BaseResultMap">
    select * from ap_author where name=#{authorName}
</select>
<insert id="insert" parameterType="com.heima.model.article.pojos.ApAuthor">
    <selectKey resultType="java.lang.Integer" order="AFTER" keyColumn="id"
keyProperty="id">
        select last_insert_id()
    </selectKey>
    insert into ap_author (name, type, user_id, wm_user_id,
    created_time)
    values (#{name}, #{type}, #{userId},#{wmUserId}, #{createdTime})
</insert>
```

修改com.heima.model.mappers.app.ApArticleContentMapper，添加新增的方法

```
void insert(ApArticleContent apArticleContent);
```

ApArticleContentMapper.xml

```
<insert id="insert"
parameterType="com.heima.model.article.pojos.ApArticleContent">
    insert into ap_article_content (article_id,content) values (#{articleId},#
{content})
</insert>
```

修改com.heima.model.mappers.app.ApArticleConfigMapper，添加新增数据的方法

```
int insert(ApArticleConfig apArticleConfig);
```

ApArticleConfigMapper.xml

```
<insert id="insert"
parameterType="com.heima.model.article.pojos.ApArticleConfig">
    insert into ap_article_config
(article_id,is_comment,is_forward,is_down,is_delete) values(
    #{articleId},#{isComment},#{isForward},#{isDown},#{isDelete}
    );
</insert>
```

修改com.heima.model.mappers.app.ApArticleMapper，添加新增数据的方法

```
void insert(ApArticle apArticle);
```

ApArticleMapper.xml

```
<insert id="insert" parameterType="com.heima.model.article.pojos.ApArticle">
    <selectKey resultType="java.lang.Integer" order="AFTER" keyColumn="id"
keyProperty="id">
        select last_insert_id()
    </selectKey>
```

```
    insert into ap_article
    (title,
    author_id,
    author_name,
    channel_id,
    channel_name,
    layout,
    flag,
    images,
    labels,
    likes,
    collection,
    comment,
    views,
    province_id,
    city_id,
    county_id,
    created_time,
    publish_time,sync_status,origin)
    values
    (#{title},
    #{authorId},
    #{authorName},
    #{channelId},
    #{channelName},
    #{layout},
    #{flag},
    #{images},
    #{labels},
    #{likes},
    #{collection},
    #{comment},
    #{views},
    #{provinceId},
    #{cityId},
    #{countyId},
    #{createdTime},
    #{publishTime},
    #{syncStatus},
    #{origin}
    )
</insert>
```

修改com.heima.model.mappers.wemedia.WmNewsMapper，添加修改的方法

```
int updateByPrimaryKeySelective(WmNews record);
```

WmNewsMapper.xml

```
<update id="updateByPrimaryKeySelective"
parameterType="com.heima.model.media.pojos.WmNews" >

    update wm_news
    <set >
        <if test="userId != null" >
            user_id = #{userId},
```

```
        </if>
        <if test="title != null" >
            title = #{title},
        </if>
        <if test="type != null" >
            type = #{type},
        </if>
        <if test="channelId != null" >
            channel_id = #{channelId},
        </if>
        <if test="labels != null" >
            labels = #{labels},
        </if>
        <if test="createdTime != null" >
            created_time = #{createdTime},
        </if>
        <if test="submitedTime != null" >
            submited_time = #{submitedTime},
        </if>
        <if test="status != null" >
            status = #{status,jdbcType=TINYINT},
        </if>
        <if test="publishTime != null" >
            publish_time = #{publishTime},
        </if>
        <if test="reason != null" >
            reason = #{reason},
        </if>
        <if test="articleId != null" >
            article_id = #{articleId},
        </if>
        <if test="content != null" >
            content = #{content,jdbcType=LONGVARCHAR},
        </if>
    </set>
    where id = #{id}
</update>
```

新建类：com.heima.model.mappers.app.ApUserMessageMapper

```
int insertSelective(ApUserMessage record);
```

ApUserMessageMapper.xml

```
<resultMap id="BaseResultMap" type="com.heima.model.user.pojos.ApUserMessage" >
    <result column="id" property="id"/>
    <result column="user_id" property="userId"/>
    <result column="sender_id" property="senderId"/>
    <result column="sender_name" property="senderName"/>
    <result column="content" property="content"/>
    <result column="type" property="type"/>
    <result column="is_read" property="isRead"/>
    <result column="created_time"  property="createdTime"/>
    <result column="read_time"  property="readTime"/>
</resultMap>
```

```
<insert id="insertSelective"
parameterType="com.heima.model.user.pojos.ApUserMessage" >
    insert into ap_user_message
    <trim prefix="(" suffix=")" suffixOverrides="," >
        <if test="id != null" >
            id,
        </if>
        <if test="userId != null" >
            user_id,
        </if>
        <if test="senderId != null" >
            sender_id,
        </if>
        <if test="senderName != null" >
            sender_name,
        </if>
        <if test="content != null" >
            content,
        </if>
        <if test="type != null" >
            type,
        </if>
        <if test="isRead != null" >
            is_read,
        </if>
        <if test="createdTime != null" >
            created_time,
        </if>
        <if test="readTime != null" >
            read_time,
        </if>
    </trim>
    <trim prefix="values (" suffix=")" suffixOverrides="," >
        <if test="id != null" >
            #{id},
        </if>
        <if test="userId != null" >
            #{userId},
        </if>
        <if test="senderId != null" >
            #{senderId},
        </if>
        <if test="senderName != null" >
            #{senderName},
        </if>
        <if test="content != null" >
            #{content},
        </if>
        <if test="type != null" >
            #{type,jdbcType=TINYINT},
        </if>
        <if test="isRead != null" >
            #{isRead},
        </if>
        <if test="createdTime != null" >
            #{createdTime},
        </if>
        <if test="readTime != null" >
```

```
            #{readTime},
        </if>
    </trim>
</insert>
```

（2）service接口com.heima.admin.service.ReviewMediaArticleService

```
public interface ReviewMediaArticleService {

    /**
     * 自媒体端发布文章审核
     * @param newsId 文章id
     */
    public void autoReviewArticleByMedia(Integer newsId);
}
```

（3）service实现类com.heima.admin.service.impl.ReviewMediaArticleServiceImpl

```
@Service
@Log4j2
@SuppressWarnings("all")
public class ReviewMediaArticleServiceImpl implements ReviewMediaArticleService
{
    @Autowired
    private WmNewsMapper wmNewsMapper;

    @Value("${review_article_pass}")
    private Double review_article_pass;

    @Value("${review_article_review}")
    private Double review_article_review;

    @Autowired
    private AliyunTextScanRequest aliyunTextScanRequest;

    @Autowired
    private AliyunImageScanRequest aliyunImageScanRequest;

    @Autowired
    private JestClient jestClient;

    //ap_article_config   ap_article   ap_article_content  ap_author
    @Autowired
    private ApArticleConfigMapper apArticleConfigMapper;

    @Autowired
    private ApArticleMapper apArticleMapper;

    @Autowired
    private ApArticleContentMapper apArticleContentMapper;

    @Autowired
    private ApAuthorMapper apAuthorMapper;

    @Autowired
    private AdChannelMapper adChannelMapper;
```

```
    @Autowired
    private WmUserMapper wmUserMapper;

    @Autowired
    private ApUserMessageMapper apUserMessageMapper;

    @Autowired
    KafkaSender kafkaSender;
    /**
     * 主图数量
     */
    private static final Integer MAIN_PICTURE_SIZE = 5;

    @Value("${FILE_SERVER_URL}")
    private String FILE_SERVER_URL;

    /**
     * 自媒体文章发布审核
     *
     * @param newsId 文章id
     */
    @Override
    public void autoReviewArticleByMedia(Integer newsId) {
        //1.根据文章id查询文章内容，内容包括文本和图片
//        WmNews wmNews = wmNewsMapper.findbyNewsIdAndStatus(newsId, 1);
        WmNews wmNews = wmNewsMapper.selectByPrimaryKey(newsId);

        //人工审核    直接保存数据和创建索引
        if (wmNews != null && wmNews.getStatus() == 4) {
            reviewSuccessSaveAll(wmNews, wmNews.getContent(),
wmNews.getTitle());
            return;
        }
        //审核通过后待发布的文章，判断发布时间
        if (wmNews != null && wmNews.getStatus() == 8 &&
wmNews.getPublishTime().getTime() > new Date().getTime()) {
            reviewSuccessSaveAll(wmNews, wmNews.getContent(),
wmNews.getTitle());
            return;
        }

        if (wmNews != null && wmNews.getStatus() == 1) {
            //2.根据文章标题匹配文章内容 匹配度
            String content = wmNews.getContent();//文章内容
            String title = wmNews.getTitle();//文章标题
            //TODO 测试阶段，暂时不对文章标题和文章内容匹配的审核
        /*double semblance = SimHashUtils.getSemblance(title, content, 64);
        if(semblance<review_article_pass && semblance>review_article_review){
            //人工审核，修改状态
            updateWmNews(wmNews, (short) 3, "人工复审");
            return;
        }
        if(semblance<review_article_review){
            //文章与标题不匹配，拒绝
            updateWmNews(wmNews, (short) 2, "文章与标题不匹配");

            //系统推送消息，审核结果
```

```
            saveApUserMessage(wmNews, 109, "审核未通过，文章内容与标题不匹配");
            return;
        }*/


            //3.调用阿里接口，审核文本
            List<String> images = new ArrayList<>();
            StringBuilder sb = new StringBuilder();
            JSONArray jsonArray = JSON.parseArray(content);
            handlerTextAndImages(images, sb, jsonArray);
            //截取之前五张图片
            if (images.size() > MAIN_PICTURE_SIZE) {
                images = images.subList(0, MAIN_PICTURE_SIZE);
            }


            //TODO 测试阶段 暂时不进行图片和文本内容的审核
            /*String response =
aliyunTextScanRequest.textScanRequest(sb.toString());
            if(!"pass".equals(response)){
                //文章内容审核没有通过
                updateWmNews(wmNews, (short) 2, "文章内容有违规行为");
                return;
            }
            //4.调用阿里云AI  审核图片

           response = aliyunImageScanRequest.imageScanRequest(images);
            if(response ==null || !"pass".equals(response)){
                //文章图片审核没有通过
                updateWmNews(wmNews, (short) 2, "文章图片有违规行为");
                return;
            }*/
            //审核通过以后查看发布的时间，如果发布时间大于当前时间则正常发布保存时间，否则修改
状态为待发布
            if (wmNews.getPublishTime() != null) {
                if (wmNews.getPublishTime().getTime() >
System.currentTimeMillis()) {
                    // 定时发布
                    updateWmNews(wmNews, (short) 8, "待发布");
                } else {
                    // 立即发布
                    reviewSuccessSaveAll(wmNews, content, title);
                }
            } else {
                // 立即发布
                reviewSuccessSaveAll(wmNews, content, title);
            }
        }


    }

    /**
     * 审核通过保存全部信息
     */
    private void reviewSuccessSaveAll(WmNews wmNews, String content, String
title) {
        //5.如果全部通过，插入数据，同时创建索引库
```

```
        //存入数据   ap_article_config   ap_article   ap_article_content
ap_author
        Integer channelId = wmNews.getChannelId();
        AdChannel apUserChannel = adChannelMapper.selectByPrimaryKey(channelId);
        String channelName = "";
        if (apUserChannel != null) {
            channelName = apUserChannel.getName();
        }

        Date createdTime = wmNews.getCreatedTime();
        WmUser wmUser = wmUserMapper.selectById(wmNews.getUserId());
        String authorName = "";
        if (wmUser != null) {
            authorName = wmUser.getName();
        }

        String reason = wmNews.getReason();
        Short type = wmNews.getType();

        //APP文章作者信息表
//            ApAuthor apAuthor = saveApAuthor(createdTime, wmUser, authorName);
        ApAuthor apAuthor = apAuthorMapper.selectByAuthorName(authorName);
        if (apAuthor == null || apAuthor.getId() == null) {
            apAuthor = new ApAuthor();
            apAuthor.setCreatedTime(createdTime);
            apAuthor.setName(authorName);
            apAuthor.setType(1);
            apAuthor.setUserId(wmUser.getApUserId());
            apAuthor.setWmUserId(Long.valueOf(wmUser.getId()));
            apAuthorMapper.insert(apAuthor);
        }

        //文章信息表，存储已发布的文章
        ApArticle apArticle = saveApArticle(title, wmNews.getImages(),
channelId, channelName, createdTime, wmUser, authorName, apAuthor.getId(),
type);

        //APP已发布文章配置表
        saveApArticleConfig(apArticle);

        //APP已发布文章内容表(内容加密)
        saveApArticleContent(ZipUtils.gzip(content), apArticle);

        //6.创建索引
        //channelId   content   id   pub_time   publishTime   status   title   userId
tag
        EsIndexEntity esIndexEntity = saveIndexEntity(wmNews, content, title,
channelId, apArticle);

        Index.Builder builder = new Index.Builder(esIndexEntity);

        builder.id(apArticle.getId().toString());
        builder.refresh(true);
        Index index =
builder.index(ESIndexConstants.ARTICLE_INDEX).type(ESIndexConstants.DEFAULT_DOC)
.build();
```

```
        JestResult result = null;
        try {
            result = jestClient.execute(index);
        } catch (IOException e) {
            log.error("执行ES创建索引失败：message:{}", e.getMessage());
        }
        if (result != null && !result.isSucceeded()) {
            //throw new RuntimeException(result.getErrorMessage() + "插入更新索引
失败!");
            log.error("插入更新索引失败：message:{}", result.getErrorMessage());
        }

        //修改wmNews的状态为9
        wmNews.setArticleId(apArticle.getId());
        updateWmNews(wmNews, (short) 9, "审核成功");
        //通知用户 文章审核通过
        saveApUserMessage(wmNews, 108, "文章审核通过");
        try {
            // 发送热数据处理消息,此过程不影响审核结果
            ArticleAuditSuccess articleAuditSuccess = new ArticleAuditSuccess();
            articleAuditSuccess.setArticleId(apArticle.getId());
            articleAuditSuccess.setChannelId(apArticle.getChannelId());

 articleAuditSuccess.setType(ArticleAuditSuccess.ArticleType.WEMEDIA);
            kafkaSender.sendArticleAuditSuccessMessage(articleAuditSuccess);
        } catch (Exception e) {
            log.error("自动审核发送消息时错误：", e);
        }
    }

    /**
     * 创建索引
     *
     * @param wmNews
     * @param content
     * @param title
     * @param channelId
     * @param apArticle
     * @return
     */
    private EsIndexEntity saveIndexEntity(WmNews wmNews, String content, String
title, Integer channelId, ApArticle apArticle) {
        EsIndexEntity esIndexEntity = new EsIndexEntity();
        esIndexEntity.setId(new Long(apArticle.getId()));
        esIndexEntity.setChannelId(new Long(channelId));
        esIndexEntity.setContent(content);
        esIndexEntity.setPublishTime(new Date());
        esIndexEntity.setStatus(new Long(1));
        esIndexEntity.setTitle(title);
        esIndexEntity.setUserId(wmNews.getUserId());
        esIndexEntity.setTag(AdminConstans.ES_INDEX_TAG_ARTICLE);
        return esIndexEntity;
    }

    /**
     * 创建用户
     *
     * @param createdTime
```

```
     * @param wmUser
     * @param authorName
     */
    private ApAuthor saveApAuthor(Date createdTime, WmUser wmUser, String
authorName) {
        ApAuthor apAuthor = new ApAuthor();
        apAuthor.setCreatedTime(createdTime);
        apAuthor.setUserId(wmUser.getId());
        apAuthor.setName(authorName);
        apAuthor.setType(2);
        apAuthorMapper.insert(apAuthor);

        return apAuthor;
    }

    /**
     * 保存文章内容
     *
     * @param content
     * @param apArticle
     */
    private void saveApArticleContent(String content, ApArticle apArticle) {
        ApArticleContent apArticleContent = new ApArticleContent();
        apArticleContent.setArticleId(apArticle.getId());
        apArticleContent.setContent(content);
        apArticleContentMapper.insert(apArticleContent);
    }

    /**
     * 保存文章配置信息
     *
     * @param apArticle
     */
    private void saveApArticleConfig(ApArticle apArticle) {
        ApArticleConfig apArticleConfig = new ApArticleConfig();
        apArticleConfig.setArticleId(apArticle.getId());
        apArticleConfig.setIsComment(true);
        apArticleConfig.setIsDelete(false);
        apArticleConfig.setIsDown(false);
        apArticleConfig.setIsForward(true);
        apArticleConfigMapper.insert(apArticleConfig);
    }

    /**
     * 保存文章信息
     *
     * @param title
     * @param images
     * @param channelId
     * @param channelName
     * @param createdTime
     * @param publishTime
     * @param wmUser
     * @param authorName
     * @return
     */
```

```
    private ApArticle saveApArticle(String title, String images, Integer
channelId, String channelName, Date createdTime, WmUser wmUser, String
authorName, Integer authorId, Short type) {
        ApArticle apArticle = new ApArticle();
        apArticle.setChannelId(channelId);
        apArticle.setChannelName(channelName);
        apArticle.setAuthorId(wmUser.getId());
        apArticle.setAuthorName(authorName);
        apArticle.setCreatedTime(createdTime);
        apArticle.setOrigin(true);
        if (images != null) {
            String[] split = images.split(",");
            StringBuilder sb = new StringBuilder();
            for (int i = 0; i < split.length; i++) {
                if (i != 0) {
                    sb.append(",");
                }
                sb.append(FILE_SERVER_URL);
                sb.append(split[i]);
            }
            apArticle.setImages(sb.toString());
        }
        apArticle.setLayout(type);
        apArticle.setTitle(title);
        apArticle.setPublishTime(new Date());
        apArticle.setAuthorId(new Long(authorId));

        apArticleMapper.insert(apArticle);
        return apArticle;
    }

    /**
     * 处理content  找出文本和图片列表
     *
     * @param images
     * @param sb
     * @param jsonArray
     */
    private void handlerTextAndImages(List<String> images, StringBuilder sb,
JSONArray jsonArray) {
        for (Object obj : jsonArray) {
            JSONObject jsonObj = (JSONObject) obj;
            String type = (String) jsonObj.get("type");
            if ("image".equals(type)) {
                String value = (String) jsonObj.get("value");
                images.add(value);
            }
            if ("text".equals(type)) {
                sb.append(jsonObj.get("value"));
            }
        }
    }

    /**
     * 审核失败  更新状态及失败说明
     */
    private void updateWmNews(WmNews wmNews, short i, String message) {
        wmNews.setStatus(i);
```

```
        wmNews.setReason(message);
        wmNewsMapper.updateByPrimaryKeySelective(wmNews);
    }

    /**
     * 审核失败 在用户的通知消息表中存入消息，告知用户
     *
     * @param wmNews
     * @param i
     * @param s
     */
    private void saveApUserMessage(WmNews wmNews, int i, String s) {
        ApUserMessage apUserMessage = new ApUserMessage();
        apUserMessage.setUserId(wmNews.getUserId());
        apUserMessage.setCreatedTime(new Date());
        apUserMessage.setIsRead(false);
        apUserMessage.setType(i);
        apUserMessage.setContent(s);
        apUserMessageMapper.insertSelective(apUserMessage);
    }

}
```

2.2.4 单元测试

```
@SpringBootTest
@RunWith(SpringRunner.class)
public class ReviewArticleTest {

    @Autowired
    private ReviewMediaArticleService reviewMediaArticleService;

    @Test
    public void testReview(){
        reviewMediaArticleService.autoReviewArticleByMedia(6110);
    }

}
```

2.2.4 自媒体发布文章成功后发送消息

（1）定义消息

maven_test.properties

```
kafka.topic.submit-article-auth=heima.topic.submit.article.auth.sigle.test
```

kafka.properties

```
kafka.topic.submit-article-auth=${kafka.topic.submit-article-auth}
```

修改com.heima.common.kafka.KafkaTopicConfig,添加属性

```
String submitArticleAuth;
```

（2）创建包装类com.heima.model.mess.admin.SubmitArticleAuto,主要功能是封装传递的消息，并且区分消息的类型

```
@Data
public class SubmitArticleAuto {

    // 文章类型
    private ArticleType type;
    // 文章ID
    private Integer articleId;

    public enum ArticleType{
        WEMEDIA,CRAWLER;
    }
}
```

（3）创建类com.heima.common.kafka.messages.admin.SubmitArticleAuthMessage,继承KafkaMessage，主要功能是再做一次封装，可以针对单通道和多通道进行区分

```
public class SubmitArticleAuthMessage extends KafkaMessage<SubmitArticleAuto> {

    public SubmitArticleAuthMessage(){}

    public SubmitArticleAuthMessage(SubmitArticleAuto data){
        super(data);
    }

    @Override
    public String getType() {
        return "submit-article-auth";
    }
}
```

（4）修改com.heima.common.kafka.KafkaSender添加方法

```
/**
     * 发送审核文章的消息
     *
     * @param message
     */
public void sendSubmitArticleAuthMessage(SubmitArticleAuthMessage message) {
    this.sendMesssage(kafkaTopicConfig.getSubmitArticleAuth(),
UUID.randomUUID().toString(), message);
}
```

（5）在media端创建类：com.heima.media.kafka.AdminMessageSender,用来发送消息

```
@Component
public class AdminMessageSender {

    @Autowired
    KafkaSender kafkaSender;

    /**
     * 只发送行为消息
```

```
     * @param message
     */
    @Async
    public void sendMessage(SubmitArticleAuthMessage message){
        kafkaSender.sendSubmitArticleAuthMessage(message);
    }
}
```

（6）修改com.heima.media.service.impl.NewsServiceImpl中的saveWmNews方法，在最下面添加发送消息的代码

在该类中注入AdminMessageSender

```
// 提交才进行发送消息
if(temp==1&&WmMediaConstans.WM_NEWS_SUMMIT_STATUS==type){
    SubmitArticleAuto saa = new SubmitArticleAuto();
    saa.setArticleId(wmNews.getId());
    saa.setType(SubmitArticleAuto.ArticleType.WEMEDIA);
    adminMessageSender.sendMessage(new SubmitArticleAuthMessage(saa));
}
```

2.2.5 自媒体审核接收消息，审核文章

```
@Component
@Log4j2
public class AutoReviewArticleListener implements KafkaListener<String,String> {

    @Autowired
    private KafkaTopicConfig kafkaTopicConfig;

    @Autowired
    private ObjectMapper mapper;

    @Autowired
    private ReviewMediaArticleService reviewMediaArticleService;

    @Override
    public String topic() {
        return kafkaTopicConfig.getSubmitArticleAuth();
    }

    @Override
    public void onMessage(ConsumerRecord<String, String> consumerRecord,
Consumer<?, ?> consumer) {
        String value = consumerRecord.value();
        log.info("接收到的消息为：{}"+value);
        try {
            SubmitArticleAuthMessage message = mapper.readValue(value,
SubmitArticleAuthMessage.class);
            if(message!=null){
                SubmitArticleAuto.ArticleType type =
message.getData().getType();
                if(type==SubmitArticleAuto.ArticleType.WEMEDIA){
                    Integer articleId = message.getData().getArticleId();
```

```
                    if(articleId!=null){
                        //审核文章信息

 reviewMediaArticleService.autoReviewArticleByMedia(articleId);
                    }
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
            log.error("处理自动审核文章错误:[{}],{}",value,e);
            throw new RuntimeException("WS消息处理错误",e);
        }
    }
}
```