热点文章处理

目标

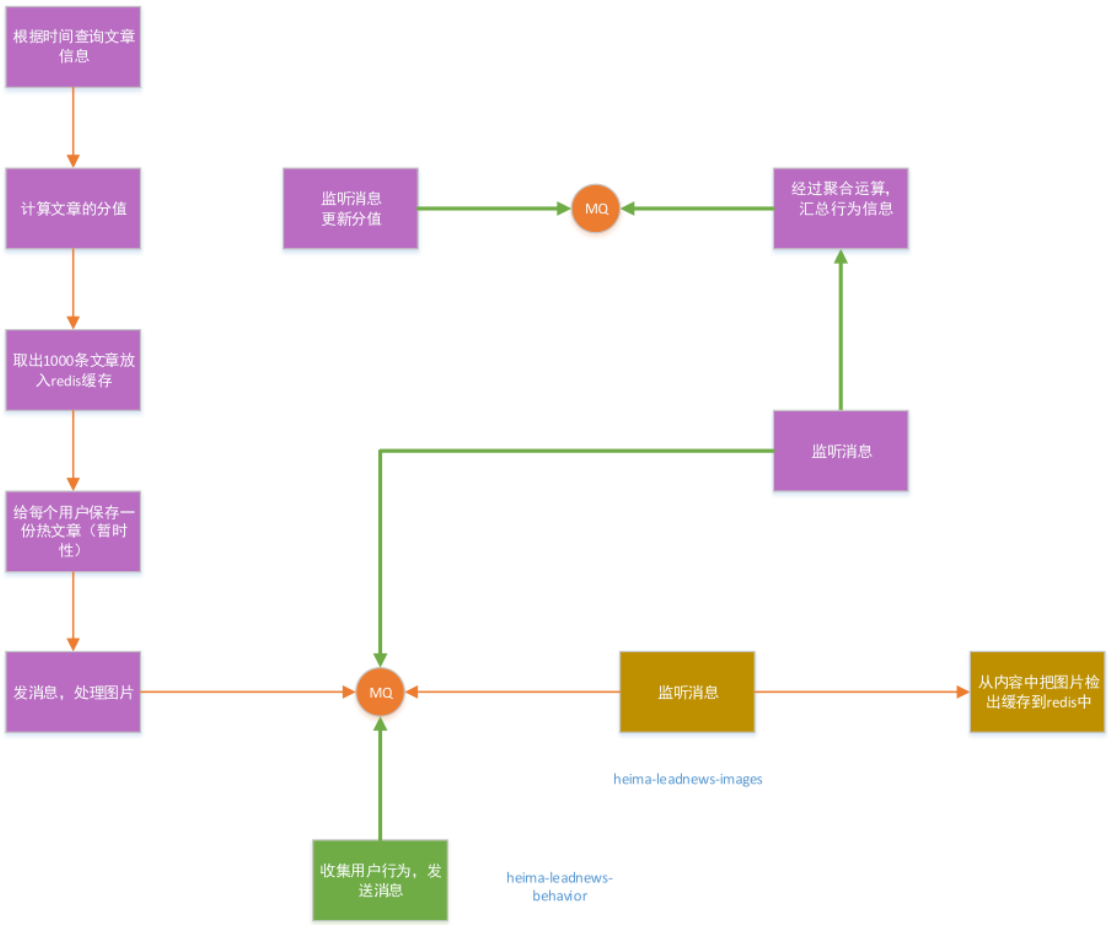
- 能够清晰计算热文章数据的流程
- 能够编写定时定时计算热文章的功能
- 能够编写实时计算文章行为数据
- 了解kafkastream的应用

1 热文章处理-功能需求

为了能够更好的把文章提供给用户，推出热文章计算功能，从两个维度计算热文章数据：一是从数据库定时任务计算，二是从流实时接收按照时间窗口计算。计算结果存储到DB，供用户查询。

- 数据库计算定时任务
- 流时间窗口计算
- 用户查询热文章接口改造

核心流程请查看资料文件夹中：热文章计算流程.pdf



2 热文章处理-热数据计算

2.1 思路分析

- 每天早上00:05分查询前一天的新发布的文章列表
- 计算文章热度（阅读量，评论，点赞，收藏）
- 缓存文章中的频道
- 给每一个用户保存一份热点文章
- 缓存文章中的图片（暂时不做）

2.2 实体类

com.heima.model.article.pojos.ApHotArticles

```
@Data
public class ApHotArticles {
    private Integer id;
    private Integer entryId;
    private Integer tagId;
    private String tagName;
    private Integer score;
    private Integer articleId;
    private Date releaseDate;
    private Date createdTime;
    private Integer provinceId;
    private Integer cityId;
    private Integer countyId;
    private Integer isRead;
}
```

2.3 Mapper实现

（1）ApHotArticlesMapper

创建类com.heima.model.mappers.app.ApHotArticlesMapper

```
public interface ApHotArticlesMapper {

    /**
     * 插入热文章数据
     *
     * @param record
     * @return
     */
    int insert(ApHotArticles record);

    /**
     * 移除传入日期之前的文章
     *
     * @param removeDate
     * @return
     */
    int removeHotArticle(String removeDate);
```

```
    /**
     * 根据ID删除
     *
     * @param id
     */
    void deleteById(Integer id);

    /**
     * 查询热文章ID
     *
     * @param entryId
     * @param dto
     * @return
     */
    List<ApHotArticles> loadArticleIdListByEntryId(Integer entryId,
ArticleHomeDto dto, short type);


    List<ApHotArticles> selectList(ApHotArticles apHotArticles);


    List<ApHotArticles> selectExpireMonth();

    /**
     * 查询今天最大ID
     * @param today
     * @return
     */
    Integer selectTodayMaxScore(String today);

    /**
     * 文章已经阅读
     * @param entryId
     * @param articleId
     * @return
     */
    int updateReadStatus(Integer entryId, Integer articleId);

    List<ApHotArticles> loadHotListByLocation(@Param("dto") ArticleHomeDto dto,
short type);
}
```

ApHotArticlesMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.heima.model.mappers.app.ApHotArticlesMapper">

    <resultMap id="BaseResultMap"
type="com.heima.model.article.pojos.ApHotArticles">
        <id column="id" property="id"/>
        <result column="entry_id" property="entryId"/>
        <result column="tag_id" property="tagId"/>
        <result column="tag_name" property="tagName"/>
        <result column="score" property="score"/>
```

```
        <result column="article_id" property="articleId"/>
        <result column="province_id" property="provinceId"/>
        <result column="city_id" property="cityId"/>
        <result column="county_id" property="countyId"/>
        <result column="is_read" property="isRead"/>
        <result column="release_date" property="releaseDate"/>
        <result column="created_time" property="createdTime"/>
    </resultMap>


    <sql id="Base_Column_List">
    id, entry_id, tag_id, tag_name, score, article_id,
    province_id,city_id,county_id,is_read,release_date,
    created_time
  </sql>

    <sql id="Base_Column_Where">
        <where>
            <if test="entryId!=null and entryId!=''">
                and entry_id = #{entryId}
            </if>
            <if test="tagId!=null and tagId!=''">
                and tag_id = #{tagId}
            </if>
            <if test="tagName!=null and tagName!=''">
                and tag_name = #{tagName}
            </if>
            <if test="score!=null and score!=''">
                and score = #{score}
            </if>
            <if test="articleId!=null and articleId!=''">
                and article_id = #{articleId}
            </if>
        </where>
    </sql>


    <select id="selectList" resultMap="BaseResultMap">
        select
        <include refid="Base_Column_List"/>
        from ap_hot_articles
        <include refid="Base_Column_Where"/>
    </select>


    <select id="selectExpireMonth" resultMap="BaseResultMap">
        select
        <include refid="Base_Column_List"/>
        from ap_hot_articles
        <where>
            where created_time > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
        </where>
    </select>

    <insert id="insert"
parameterType="com.heima.model.article.pojos.ApHotArticles">
        insert into ap_hot_articles
        <trim prefix="(" suffix=")" suffixOverrides=",">
```

```
            <if test="id != null">
                id,
            </if>
            <if test="entryId != null">
                entry_id,
            </if>
            <if test="tagId != null">
                tag_id,
            </if>
            <if test="tagName != null">
                tag_name,
            </if>
            <if test="score != null">
                score,
            </if>
            <if test="articleId != null">
                article_id,
            </if>
            <if test="provinceId != null">
                province_id,
            </if>
            <if test="cityId != null">
                city_id,
            </if>
            <if test="countyId != null">
                county_id,
            </if>
            <if test="isRead != null">
                is_read,
            </if>
            <if test="releaseDate != null">
                release_date,
            </if>
            <if test="createdTime != null">
                created_time,
            </if>
        </trim>
        <trim prefix="values (" suffix=")" suffixOverrides=",">
            <if test="id != null">
                #{id,jdbcType=INTEGER},
            </if>
            <if test="entryId != null">
                #{entryId,jdbcType=INTEGER},
            </if>
            <if test="tagId != null">
                #{tagId,jdbcType=INTEGER},
            </if>
            <if test="tagName != null">
                #{tagName,jdbcType=VARCHAR},
            </if>
            <if test="score != null">
                #{score,jdbcType=INTEGER},
            </if>
            <if test="articleId != null">
                #{articleId,jdbcType=INTEGER},
            </if>
            <if test="provinceId != null">
                #{provinceId,jdbcType=INTEGER},
```

```
            </if>
            <if test="cityId != null">
                #{cityId,jdbcType=INTEGER},
            </if>
            <if test="countyId != null">
                #{countyId,jdbcType=INTEGER},
            </if>
            <if test="isRead != null">
                #{isRead,jdbcType=TINYINT},
            </if>
            <if test="releaseDate != null">
                #{releaseDate,jdbcType=TIMESTAMP},
            </if>
            <if test="createdTime != null">
                #{createdTime,jdbcType=TIMESTAMP},
            </if>
        </trim>
    </insert>
    <!--
    删除热文章数据
  -->
    <delete id="removeHotArticle" parameterType="java.lang.String">
    delete from ap_hot_articles
    where created_time &lt; #{removeDate}
  </delete>

    <delete id="deleteById" parameterType="java.lang.Integer">
         delete from ap_hot_articles where id= #{id}
    </delete>

    <!--
      查询热文章数据
    -->
    <select id="loadArticleIdListByEntryId" parameterType="map"
resultMap="BaseResultMap">
        select * from ap_hot_articles
        <where>
            entry_id = #{entryId}  and is_read=0
            <!-- loadmore -->
            <if test="type != null and type == 1">
                and release_date <![CDATA[<]]> #{dto.minBehotTime}
            </if>
            <if test="type != null and type == 2">
                and release_date <![CDATA[>]]> #{dto.maxBehotTime}
            </if>
            <if test="dto.tag != '__all__'">
                and tag_id = #{dto.tag}
            </if>
        </where>
        limit #{dto.size}
    </select>

    <select id="selectTodayMaxScore" resultType="java.lang.Integer">
    select max(score) from ap_hot_articles
    where created_time &gt; #{today}
  </select>

    <update id="updateReadStatus">
```

```
      update ap_hot_articles set is_read = 1
      where entry_id = #{entryId} and article_id = #{articleId}
  </update>

    <!-- 依据地理位置获取 -->
    <select id="loadHotListByLocation" resultMap="BaseResultMap">
        select * from ap_hot_articles a
        <where>
            a.entry_id = 0
            <if test="dto.provinceId!=null">
                and a.province_id=#{dto.provinceId}
            </if>
            <if test="dto.cityId!=null">
                and a.city_id=#{dto.cityId}
            </if>
            <if test="dto.countyId!=null">
                and a.county_id=#{dto.countyId}
            </if>
            <!-- loadmore -->
            <if test="type != null and type == 1">
                and a.release_date <![CDATA[<]]> #{dto.minBehotTime}
            </if>
            <if test="type != null and type == 2">
                and a.release_date <![CDATA[>]]> #{dto.maxBehotTime}
            </if>
            <if test="dto.tag != '__all__'">
                and a.tag_id = #{dto.tag}
            </if>
        </where>
        limit #{dto.size}
    </select>
</mapper>
```

(2)修改com.heima.article.mysql.core.model.mappers.app.ApArticleMapper，添加根据发布时间查询的方法

```
/**
     * 抽取最近的文章数据用于计算热文章
     *
     * @param lastDate
     * @return
     */
List<ApArticle> loadLastArticleForHot(String lastDate);
```

ApArticleMapper.xml

```
<select id="loadLastArticleForHot" resultMap="resultMap">
    select
    <include refid="Base_Column_List"/>
    from ap_article
    where publish_time &gt; #{lastDate}
</select>
```

(3)修改com.heima.model.mappers.app.ApBehaviorEntryMapper 添加方法

```
List<ApBehaviorEntry> selectAllEntry();
```

ApBehaviorEntryMapper .xml

```
<!-- 查询所有 -->
<select id="selectAllEntry" resultMap="BaseResultMap" >
    select * from ap_behavior_entry
</select>
```

2.4 service代码实现

(1)AppHotArticleService

创建类：com.heima.article.service.AppHotArticleService

定义热文章计算接口：

```
public interface AppHotArticleService {

    /**
     * 计算热文章
     */
    public void computeHotArticle();
}
```

(2)AppHotArticleServiceImpl

创建类：com.heima.login.service.AppHotArticleServiceImpl

```
@Service
@SuppressWarnings("all")
public class AppHotArticleServiceImpl implements AppHotArticleService {
    @Autowired
    private ApHotArticlesMapper apHotArticlesMapper;

    @Autowired
    private ApArticleMapper apArticleMapper;

    @Autowired
    private ApBehaviorEntryMapper apBehaviorEntryMapper;

    @Autowired
    private AdChannelMapper adChannelMapper;
    @Autowired
    private StringRedisTemplate redisTemplate;

    @Override
    public void computeHotArticle() {
        //计算逻辑
        String lastDay = DateTime.now().minusDays(1).toString("yyyy-MM-dd
00:00:00");
        List<ApArticle> articleList =
apArticleMapper.loadLastArticleForHot(lastDay);
        List<ApHotArticles> hotArticlesList = computeHotArticle(articleList);
```

```
        //缓存频道到redis
        cacheTagToRedis(articleList);

        //所有用户
        List<ApBehaviorEntry> entryList =
apBehaviorEntryMapper.selectAllEntry();
        for(ApHotArticles hot : hotArticlesList){
            //插入热文章数据
            apHotArticlesMapper.insert(hot);
            //为每位用户保存一份
            saveHotArticleForEntryList(hot, entryList);
            //热文章计算完成发送给图片队列
            //通知处理热文章图片
            //kafkaSender.sendHotArticleMessage(hot);
        }
    }

    /**
     * 计算热文章
     * @param articleList
     * @return
     */
    private List<ApHotArticles> computeHotArticle(List<ApArticle> articleList) {
        List<ApHotArticles> hotArticlesList = Lists.newArrayList();
        ApHotArticles hot = null;
        for (ApArticle a : articleList) {
            hot = initHotBaseApArticle(a);
            Integer score = computeScore(a);
            hot.setScore(score);
            hotArticlesList.add(hot);
        }
        hotArticlesList.sort(new Comparator<ApHotArticles>() {
            @Override
            public int compare(ApHotArticles o1, ApHotArticles o2) {
                return o1.getScore() < o2.getScore() ? 1 : -1;
            }
        });
        if(hotArticlesList.size()>1000){
            return hotArticlesList.subList(0,1000);
        }
        return hotArticlesList;
    }

    /**
     * 初始化热文章属性
     * @param article
     * @return
     */
    private ApHotArticles initHotBaseApArticle(ApArticle article){
        ApHotArticles hot = new ApHotArticles();
        hot.setEntryId(0);
        //根据articleID查询
        hot.setTagId(article.getChannelId());
        hot.setTagName(article.getChannelName());
        hot.setScore(0);
        hot.setArticleId(article.getId());
        //设置省市区
```

```
        hot.setProvinceId(article.getProvinceId());
        hot.setCityId(article.getCityId());
        hot.setCountyId(article.getCountyId());
        hot.setIsRead(0);
        //日期
        hot.setReleaseDate(article.getPublishTime());
        hot.setCreatedTime(new Date());
        return hot;
    }

    /**
     * 计算热度分规则 1.0
     * @param a
     * @return
     */
    private Integer computeScore(ApArticle a) {
        Integer score = 0;
        if(a.getLikes()!=null){
            score += a.getLikes();
        }
        if(a.getCollection()!=null){
            score += a.getCollection();
        }
        if(a.getComment()!=null){
            score += a.getComment();
        }
        if(a.getViews()!=null){
            score += a.getViews();
        }
        return score;
    }

    /**
     * 为每位用户保存一份
     * @param hot
     * @param entryList
     */
    private void saveHotArticleForEntryList(ApHotArticles hot,
List<ApBehaviorEntry> entryList) {
        for (ApBehaviorEntry entry: entryList){
            hot.setEntryId(entry.getId());
            apHotArticlesMapper.insert(hot);
        }
    }

    /**
     * 缓存频道首页到redis
     * @param articlesList
     */
    private void cacheTagToRedis(List<ApArticle> articlesList) {
        List<AdChannel> channels = adChannelMapper.selectAll();
        List<ApArticle> temp = null;
        for (AdChannel channel : channels){
            temp = articlesList.stream().
                    filter(p -> p.getChannelId().equals(channel.getId()))
                    .collect(Collectors.toList());
            if(temp.size()>30){
                temp = temp.subList(0,30);
```

```
            }
            if(temp.size()==0){

 redisTemplate.opsForValue().set(ArticleConstans.HOT_ARTICLE_FIRST_PAGE+channel.
getId(), "");
                continue;
            }

 redisTemplate.opsForValue().set(ArticleConstans.HOT_ARTICLE_FIRST_PAGE+channel.
getId(), JSON.toJSONString(temp));
        }
    }
}
```

常量添加：com.heima.common.article.constans.ArticleConstans

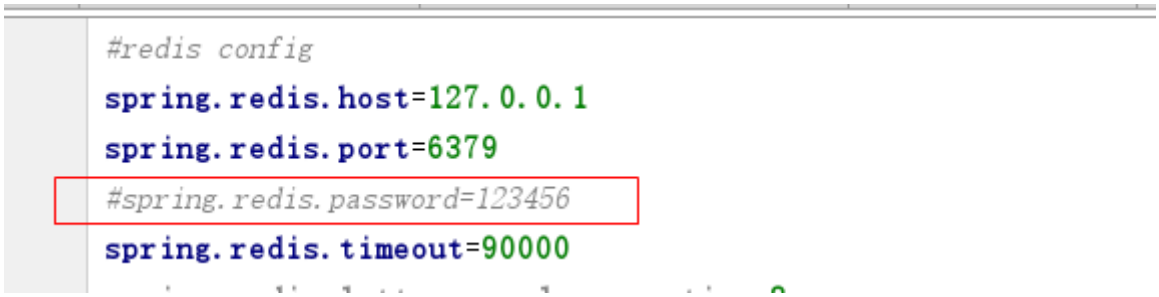
```
public class ArticleConstans {
    public static final String HOT_ARTICLE_FIRST_PAGE =
"hot_article_first_page_";
}
```

集成redis

添加类：com.heima.article.config.RedisConfig

```
@Configuration
@ComponentScan("com.heima.common.redis")
public class RedisConfig {
}
```

注意：如果你的redis访问没有密码，需要把密码项给删除



2.5 定时任务

集成quartz,注意把之前的mysql也集成到一个配置类中，并且放到quartz之前

```
@Configuration
@ComponentScan({"com.heima.common.mysql.core","com.heima.common.common.init","co
m.heima.common.quartz"})
@EnableScheduling
public class InitConfig {
}
```

AppHotArticleQuartz

创建类：com.heima.article.quartz.AppHotArticleQuartz

该类的实现较为简单，引入Service并调用即可：

```
@Component
@Log4j2
public class AppHotArticleQuartz extends AbstractJob {
    @Override
    public String[] triggerCron() {
        return new String[]{"0 0/2 * * * ?"};
    }

    @Autowired
    private AppHotArticleService appHotArticleService;

    @Override
    protected void executeInternal(JobExecutionContext jobExecutionContext)
throws JobExecutionException {
        long currentTimeMillis = System.currentTimeMillis();
        log.info("开始计算热文章数据");
        appHotArticleService.computeHotArticle();
        log.info("计算热文章数据完成，耗时:{}",System.currentTimeMillis()-
currentTimeMillis);
    }

    @Override
    public String descTrigger() {
        return "每天00:05分执行一次";
    }
}
```

创建quartz的表结构，方便集群使用定时任务

```
DROP TABLE IF EXISTS ARTICLE_QRTZ_FIRED_TRIGGERS;
DROP TABLE IF EXISTS ARTICLE_QRTZ_PAUSED_TRIGGER_GRPS;
DROP TABLE IF EXISTS ARTICLE_QRTZ_SCHEDULER_STATE;
DROP TABLE IF EXISTS ARTICLE_QRTZ_LOCKS;
DROP TABLE IF EXISTS ARTICLE_QRTZ_SIMPLE_TRIGGERS;
DROP TABLE IF EXISTS ARTICLE_QRTZ_SIMPROP_TRIGGERS;
DROP TABLE IF EXISTS ARTICLE_QRTZ_CRON_TRIGGERS;
DROP TABLE IF EXISTS ARTICLE_QRTZ_BLOB_TRIGGERS;
DROP TABLE IF EXISTS ARTICLE_QRTZ_TRIGGERS;
DROP TABLE IF EXISTS ARTICLE_QRTZ_JOB_DETAILS;
DROP TABLE IF EXISTS ARTICLE_QRTZ_CALENDARS;

CREATE TABLE ARTICLE_QRTZ_JOB_DETAILS
(
SCHED_NAME VARCHAR(120) NOT NULL,
JOB_NAME  VARCHAR(200) NOT NULL,
JOB_GROUP VARCHAR(200) NOT NULL,
DESCRIPTION VARCHAR(250) NULL,
JOB_CLASS_NAME   VARCHAR(250) NOT NULL,
IS_DURABLE VARCHAR(1) NOT NULL,
IS_NONCONCURRENT VARCHAR(1) NOT NULL,
IS_UPDATE_DATA VARCHAR(1) NOT NULL,
REQUESTS_RECOVERY VARCHAR(1) NOT NULL,
JOB_DATA BLOB NULL,
```

```
PRIMARY KEY (SCHED_NAME,JOB_NAME,JOB_GROUP)
);

CREATE TABLE ARTICLE_QRTZ_TRIGGERS
(
SCHED_NAME VARCHAR(120) NOT NULL,
TRIGGER_NAME VARCHAR(200) NOT NULL,
TRIGGER_GROUP VARCHAR(200) NOT NULL,
JOB_NAME  VARCHAR(200) NOT NULL,
JOB_GROUP VARCHAR(200) NOT NULL,
DESCRIPTION VARCHAR(250) NULL,
NEXT_FIRE_TIME BIGINT(13) NULL,
PREV_FIRE_TIME BIGINT(13) NULL,
PRIORITY INTEGER NULL,
TRIGGER_STATE VARCHAR(16) NOT NULL,
TRIGGER_TYPE VARCHAR(8) NOT NULL,
START_TIME BIGINT(13) NOT NULL,
END_TIME BIGINT(13) NULL,
CALENDAR_NAME VARCHAR(200) NULL,
MISFIRE_INSTR SMALLINT(2) NULL,
JOB_DATA BLOB NULL,
PRIMARY KEY (SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP),
FOREIGN KEY (SCHED_NAME,JOB_NAME,JOB_GROUP)
REFERENCES ARTICLE_QRTZ_JOB_DETAILS(SCHED_NAME,JOB_NAME,JOB_GROUP)
);

CREATE TABLE ARTICLE_QRTZ_SIMPLE_TRIGGERS
(
SCHED_NAME VARCHAR(120) NOT NULL,
TRIGGER_NAME VARCHAR(200) NOT NULL,
TRIGGER_GROUP VARCHAR(200) NOT NULL,
REPEAT_COUNT BIGINT(7) NOT NULL,
REPEAT_INTERVAL BIGINT(12) NOT NULL,
TIMES_TRIGGERED BIGINT(10) NOT NULL,
PRIMARY KEY (SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP),
FOREIGN KEY (SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP)
REFERENCES ARTICLE_QRTZ_TRIGGERS(SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP)
);

CREATE TABLE ARTICLE_QRTZ_CRON_TRIGGERS
(
SCHED_NAME VARCHAR(120) NOT NULL,
TRIGGER_NAME VARCHAR(200) NOT NULL,
TRIGGER_GROUP VARCHAR(200) NOT NULL,
CRON_EXPRESSION VARCHAR(200) NOT NULL,
TIME_ZONE_ID VARCHAR(80),
PRIMARY KEY (SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP),
FOREIGN KEY (SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP)
REFERENCES ARTICLE_QRTZ_TRIGGERS(SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP)
);

CREATE TABLE ARTICLE_QRTZ_SIMPROP_TRIGGERS
(
SCHED_NAME VARCHAR(120) NOT NULL,
TRIGGER_NAME VARCHAR(200) NOT NULL,
TRIGGER_GROUP VARCHAR(200) NOT NULL,
STR_PROP_1 VARCHAR(512) NULL,
STR_PROP_2 VARCHAR(512) NULL,
```

```
STR_PROP_3 VARCHAR(512) NULL,
INT_PROP_1 INT NULL,
INT_PROP_2 INT NULL,
LONG_PROP_1 BIGINT NULL,
LONG_PROP_2 BIGINT NULL,
DEC_PROP_1 NUMERIC(13,4) NULL,
DEC_PROP_2 NUMERIC(13,4) NULL,
BOOL_PROP_1 VARCHAR(1) NULL,
BOOL_PROP_2 VARCHAR(1) NULL,
PRIMARY KEY (SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP),
FOREIGN KEY (SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP)
REFERENCES ARTICLE_QRTZ_TRIGGERS(SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP)
);

CREATE TABLE ARTICLE_QRTZ_BLOB_TRIGGERS
(
SCHED_NAME VARCHAR(120) NOT NULL,
TRIGGER_NAME VARCHAR(200) NOT NULL,
TRIGGER_GROUP VARCHAR(200) NOT NULL,
BLOB_DATA BLOB NULL,
PRIMARY KEY (SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP),
FOREIGN KEY (SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP)
REFERENCES ARTICLE_QRTZ_TRIGGERS(SCHED_NAME,TRIGGER_NAME,TRIGGER_GROUP)
);

CREATE TABLE ARTICLE_QRTZ_CALENDARS
(
SCHED_NAME VARCHAR(120) NOT NULL,
CALENDAR_NAME  VARCHAR(200) NOT NULL,
CALENDAR BLOB NOT NULL,
PRIMARY KEY (SCHED_NAME,CALENDAR_NAME)
);

CREATE TABLE ARTICLE_QRTZ_PAUSED_TRIGGER_GRPS
(
SCHED_NAME VARCHAR(120) NOT NULL,
TRIGGER_GROUP  VARCHAR(200) NOT NULL,
PRIMARY KEY (SCHED_NAME,TRIGGER_GROUP)
);

CREATE TABLE ARTICLE_QRTZ_FIRED_TRIGGERS
(
SCHED_NAME VARCHAR(120) NOT NULL,
ENTRY_ID VARCHAR(95) NOT NULL,
TRIGGER_NAME VARCHAR(200) NOT NULL,
TRIGGER_GROUP VARCHAR(200) NOT NULL,
INSTANCE_NAME VARCHAR(200) NOT NULL,
FIRED_TIME BIGINT(13) NOT NULL,
SCHED_TIME BIGINT(13) NOT NULL,
PRIORITY INTEGER NOT NULL,
STATE VARCHAR(16) NOT NULL,
JOB_NAME VARCHAR(200) NULL,
JOB_GROUP VARCHAR(200) NULL,
IS_NONCONCURRENT VARCHAR(1) NULL,
REQUESTS_RECOVERY VARCHAR(1) NULL,
PRIMARY KEY (SCHED_NAME,ENTRY_ID)
);
```

```
CREATE TABLE ARTICLE_QRTZ_SCHEDULER_STATE
(
SCHED_NAME VARCHAR(120) NOT NULL,
INSTANCE_NAME VARCHAR(200) NOT NULL,
LAST_CHECKIN_TIME BIGINT(13) NOT NULL,
CHECKIN_INTERVAL BIGINT(13) NOT NULL,
PRIMARY KEY (SCHED_NAME,INSTANCE_NAME)
);

CREATE TABLE ARTICLE_QRTZ_LOCKS
(
SCHED_NAME VARCHAR(120) NOT NULL,
LOCK_NAME  VARCHAR(40) NOT NULL,
PRIMARY KEY (SCHED_NAME,LOCK_NAME)
);

commit;
```

需要配置mycat中的scheme.xml文件，配置新增的11张表

2.6 测试

启动项目测试即可

3 文章热点数据实时计算

3.1 行为采集发送消息

3.1.1 定义消息名称

maven_test.properties

```
# app产生的更新文章数据消息
kafka.topic.article-update-bus=heima.topic.app.article.update.bus.test
```

kafka.properties

```
# app产生的更新文章数据消息
kafka.topic.article-update-bus=${kafka.topic.article-update-bus}
```

在com.heima.common.kafka.KafkaTopicConfig类中新增属性，添加消息名称

```
// 更新文章数据的消息topic
String articleUpdateBus;
```

3.1.2 定时消息封装的实体类

封装具体消息的实体类

com.heima.model.mess.app.UpdateArticle

```
@Data
public class UpdateArticle implements Serializable {
```

```
    private static final long serialVersionUID = 3324988207631812651;

    // 修改文章的字段类型
    private UpdateArticleType type;
    // 文章ID
    private Integer articleId;
    // 修改数据的增量，可为正负
    private Integer add;
    private Integer apUserId;
    private Integer behaviorId;

    public enum UpdateArticleType{
        COLLECTION,COMMENT,LIKES,VIEWS;
    }
}
```

进行消息传输的实体类

com.heima.common.kafka.messages.app.UpdateArticleMessage

```
public class UpdateArticleMessage extends KafkaMessage<UpdateArticle> {

    public UpdateArticleMessage(){}

    public UpdateArticleMessage(UpdateArticle data) {
        super(data);
    }
    @Override
    public String getType() {
        return "update-article";
    }
}
```

3.1.3 添加发送消息的方法

修改com.heima.common.kafka.KafkaSender,

```
 /**
     * 发送修改文章请求消息
     *
     * @param message
     */
public void sendArticleUpdateBus(KafkaMessage message) {
    this.sendMesssage(kafkaTopicConfig.getArticleUpdateBus(),
UUID.randomUUID().toString(), message);
}
```

3.1.4用户行为消息发送类封装

在heima-leadnews-behavior模块中收集

创建消息发送类：com.heima.behavior.kafka.BehaviorMessageSender

```
@Component
public class BehaviorMessageSender {
```

```
    @Autowired
    KafkaSender kafkaSender;

    /**
     * 发送+1的消息
     * @param message
     * @param isSendToArticle
     */
    @Async
    public void sendMessagePlus(KafkaMessage message,Long apUserId,boolean
isSendToArticle){
        if(isSendToArticle){
            UpdateArticleMessage temp = parseMessage(message,apUserId,1);
            if(temp!=null)
                kafkaSender.sendArticleUpdateBus(temp);
        }
    }

    /**
     * 发送-1的消息
     * @param message
     * @param isSendToArticle
     */
    @Async
    public void sendMessageReduce(KafkaMessage message,Long apUserId,boolean
isSendToArticle){
        if(isSendToArticle){
            UpdateArticleMessage temp = parseMessage(message,apUserId,-1);
            if(temp!=null)
                kafkaSender.sendArticleUpdateBus(temp);
        }
    }

    /**
     * 转换行为消息为修改位置的消息
     * @param message
     * @param step
     * @return
     */
    private UpdateArticleMessage parseMessage(KafkaMessage message,Long
apUserId,int step){
        UpdateArticle ua = new UpdateArticle();
        if(apUserId!=null) ua.setApUserId(apUserId.intValue());
        // 转换为收藏的消息
        if(message instanceof UserCollectionMessage){
            UserCollectionMessage m = (UserCollectionMessage)message;
            // 只发收藏文章的消息
            if(m.getData().getType()== ApCollection.Type.ARTICLE.getCode()) {
                ua.setType(UpdateArticle.UpdateArticleType.COLLECTION);
                ua.setAdd(step);
                ua.setArticleId(m.getData().getEntryId());
                ua.setBehaviorId(m.getData().getBehaviorEntryId());
            }
        }else if(message instanceof UserLikesMessage){
            UserLikesMessage m = (UserLikesMessage)message;
            // 只发文章的喜欢消息
            if(m.getData().getType()== ApLikesBehavior.Type.ARTICLE.getCode()){
```

```
                ua.setType(UpdateArticle.UpdateArticleType.LIKES);
                ua.setAdd(step);
                ua.setArticleId(m.getData().getEntryId());
                ua.setBehaviorId(m.getData().getBehaviorEntryId());
            }
        }else if(message instanceof UserReadMessage){
            UserReadMessage m = (UserReadMessage) message;
            ua.setType(UpdateArticle.UpdateArticleType.VIEWS);
            ua.setAdd(step);
            ua.setArticleId(m.getData().getArticleId());
            ua.setBehaviorId(m.getData().getEntryId());
        }
        if(ua.getArticleId()!=null){
            return new UpdateArticleMessage(ua);
        }
        return null;
    }


}
```

3.1.5 具体行为消息发送

（1）点赞

创建封装点赞消息的实体类

com.heima.commom.kafka.messages.behavior.UserLikesMessage

```
public class UserLikesMessage extends KafkaMessage<ApLikesBehavior> {

    public  UserLikesMessage(){}
    public UserLikesMessage(ApLikesBehavior data) {
        super(data);
    }

    @Override
    public String getType() {
        return "user-likes";
    }
}
```

修改 com.heima.behavior.service.impl.AppLikesBehaviorServiceImpl

```
@Override
public ResponseResult saveLikesBehavior(LikesBehaviorDto dto){
    //.....上面代码省略.....
    int temp = apLikesBehaviorMapper.insert(alb);
    if(insert==1){
        if(alb.getOperation()==ApLikesBehavior.Operation.LIKE.getCode()){
            behaviorMessageSender.sendMessagePlus(new
UserLikesMessage(alb),userId,true);
        }else if(alb.getOperation()==ApLikesBehavior.Operation.CANCEL.getCode())
{
            behaviorMessageSender.sendMessageReduce(new
UserLikesMessage(alb),userId,true);
        }
    }
    return ResponseResult.okResult(temp);
}
```

（2）阅读

创建封装收藏消息的实体类

com.heima.common.kafka.messages.behavior.UserReadMessage

```
public class UserReadMessage extends KafkaMessage<ApReadBehavior> {

    public UserReadMessage(){}
    public UserReadMessage(ApReadBehavior data) {
        super(data);
    }

    @Override
    public String getType() {
        return "user-read";
    }
}
```

修改com.heima.behavior.service.impl.AppReadBehaviorServiceImpl

```
@Override
public ResponseResult saveReadBehavior(ReadBehaviorDto dto){
    //...代码省略...
    // 插入
    if(isInsert){
        int temp = apReadBehaviorMapper.insert(alb);
        if(temp==1) {
            behaviorMessageSender.sendMessagePlus(new
UserReadMessage(alb),userId, true);
        }
        return ResponseResult.okResult(temp);
    }else {
        // 更新
        return ResponseResult.okResult(apReadBehaviorMapper.update(alb));
    }
}
```

3.2 文章微服务接收消息处理

3.2.1 导入kafkastream配置类

导入资料文件中关于kafkastream相关的工具类

实现KafkaStreamListener接口的类，在KafkaStreamListenerFactory中自动扫描并实例KafkaStreamProcessor对象，并调用KafkaStreamProcessor的doAction方法，创建流计算器，并返回后注册到bean容器当中

3.2.2 定义消息名称

maven_test.properties

```
kafka.topic.article-incr-handle=kafka.topic.article.incr.handle.sigle.test
```

kafka.properties

```
kafka.topic.article-incr-handle=${kafka.topic.article-incr-handle}
```

在com.heima.common.kafka.KafkaTopicConfig类中新增属性，读取消息名称

```
//文章增量流处理完毕 处理结果监听主题
String articleIncrHandle;
```

3.2.3 定义消息处理的实体类

定义消息传递封装类：com.heima.common.kafka.messages.app.ArticleVisitStreamMessage

```
public class ArticleVisitStreamMessage extends
KafkaMessage<ArticleVisitStreamDto> {

    @Override
    public String getType() {
        return "article-visit-stream";
    }
}
```

创建包装具体消息内容的类：com.heima.model.mess.app.ArticleVisitStreamDto

```
@Data
public class ArticleVisitStreamDto {
    private Integer articleId;
    private long view;
    private long collect;
    private long commont;
    private long like;
}
```

3.2.4 定义监听类，接收流消息

创建监听类：com.heima.article.stream.HotArticleStreamHandler

```java
@Component
@Log4j2
public class HotArticleStreamHandler implements KafkaStreamListener<KStream<?,
String>> {

    @Autowired
    private KafkaTopicConfig kafkaTopicConfig;

    @Autowired
    private ObjectMapper mapper;

    @Override
    public String listenerTopic() {
        return kafkaTopicConfig.getArticleUpdateBus();
    }

    @Override
    public String sendTopic() {
        return kafkaTopicConfig.getArticleIncrHandle();
    }

    @Override
    public KStream<?, String> getService(KStream<?, String> stream) {
        return stream.map((key, val) -> {
            UpdateArticleMessage value = format(val);
            System.out.println(value);
            //likes:1
            return new KeyValue<>(value.getData().getArticleId().toString(),
value.getData().getType().name() + ":" + value.getData().getAdd());
        }).groupByKey().windowedBy(TimeWindows.of(10000)).aggregate(new
Initializer<String>() {
            @Override
            public String apply() {
                return "COLLECTION:0,COMMENT:0,LIKES:0,VIEWS:0";
            }
        }, new Aggregator<Object, String, String>() {
            @Override
            public String apply(Object aggKey, String value, String aggValue) {
                //类似于  likes:1
                value = value.replace("UpdateArticle(", "").replace(")", "");
                String valAry[] = value.split(":");
                if ("null".equals(valAry[1])) {
                    return aggValue;
                }
                //"COLLECTION:0,COMMENT:0,LIKES:0,VIEWS:0";
//                String[] aggArr = aggValue.split(",");
                int col = 0, com = 0, lik = 0, vie = 0;
                if("LIKES".equalsIgnoreCase(valAry[0])){
                    lik+=Integer.valueOf(valAry[1]);
                }
                if("COLLECTION".equalsIgnoreCase(valAry[0])){
                    col+=Integer.valueOf(valAry[1]);
                }
                if("COMMENT".equalsIgnoreCase(valAry[0])){
                    com+=Integer.valueOf(valAry[1]);
                }
                if("VIEWS".equalsIgnoreCase(valAry[0])){
                    vie+=Integer.valueOf(valAry[1]);
```

```
                }
                /*for (int i = 0; i < aggArr.length; i++) {
                    String temp[] = aggArr[i].split(":");
                    switch (UpdateArticle.UpdateArticleType.valueOf(temp[0])) {
                        case COLLECTION:
                            col = Integer.valueOf(temp[1]);
                        case COMMENT:
                            com = Integer.valueOf(temp[1]);
                        case LIKES:
                            lik = Integer.valueOf(temp[1]);
                        case VIEWS:
                            vie = Integer.valueOf(temp[1]);
                    }

                }
                switch (UpdateArticle.UpdateArticleType.valueOf(valAry[0])) {
                    case COLLECTION:
                        col += Integer.valueOf(valAry[1]);
                    case COMMENT:
                        com += Integer.valueOf(valAry[1]);
                    case LIKES:
                        lik += Integer.valueOf(valAry[1]);
                    case VIEWS:
                        vie += Integer.valueOf(valAry[1]);
                }*/
                return
String.format("COLLECTION:%d,COMMENT:%d,LIKES:%d,VIEWS:%d", col, com, lik, vie);
            }
        }, Materialized.as("count-article-num-miukoo-1")).toStream().map((key,
value) -> {
            return new KeyValue<>(key.key().toString(),
formatObj(key.key().toString(), value));
        });
    }

    private String formatObj(String articleId, String value) {
        String ret = "";
        ArticleVisitStreamMessage temp = new ArticleVisitStreamMessage();
        ArticleVisitStreamDto dto = new ArticleVisitStreamDto();
        String regEx = "COLLECTION:(\\d+),COMMENT:(\\d+),LIKES:(\\d+),VIEWS:
(\\d+)";
        Pattern pat = Pattern.compile(regEx);
        Matcher mat = pat.matcher(value);
        if (mat.find()) {
            dto.setCollect(Long.valueOf(mat.group(1)));
            dto.setCommont(Long.valueOf(mat.group(2)));
            dto.setLike(Long.valueOf(mat.group(3)));
            dto.setView(Long.valueOf(mat.group(4)));
        } else {
            dto.setCollect(0);
            dto.setCommont(0);
            dto.setLike(0);
            dto.setView(0);
        }
        dto.setArticleId(Integer.valueOf(articleId));
        temp.setData(dto);
        try {
            ret = mapper.writeValueAsString(temp);
```

```
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }
        return ret;
    }

    private UpdateArticleMessage format(String val) {
        UpdateArticleMessage msg = null;
        try {
            msg = mapper.readValue(val, UpdateArticleMessage.class);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return msg;
    }
}
```

3.2.5 更新文章增量数据

（1）mapper定义

修改com.heima.model.mappers.app.ApArticleMapper，新增方法

```
/**
     * 更新文章数
     * @param articleId
     * @param viewCount
     * @param collectCount
     * @param commontCount
     * @param likeCount
     * @return
     */
int updateArticleView(Integer articleId, long viewCount,long collectCount,long
commontCount,long likeCount);
```

ApArticleMapper.xml

```
<update id="updateArticleView">
    UPDATE ap_article SET
    views = GREATEST(IFNULL(views,0) + #{viewCount}, #{viewCount}),
    collection = GREATEST(IFNULL(collection,0) + #{collectCount}, #
{collectCount}),
    comment = GREATEST(IFNULL(comment,0) + #{commontCount}, #{commontCount}),
    likes = GREATEST(IFNULL(likes,0) + #{likeCount}, #{likeCount})
    WHERE id=#{articleId}
</update>
```

（2）service定义

在com.heima.article.service.AppArticleService新增修改方法

```
/**
     * 更新阅读数
     * @param dto
     * @return
     */
ResponseResult updateArticleView(ArticleVisitStreamDto dto);
```

实现类

```
 @Override
public ResponseResult updateArticleView(ArticleVisitStreamDto dto) {
    int rows = apArticleMapper.updateArticleView(dto.getArticleId(),

dto.getView(),dto.getCollect(),dto.getCommont(),dto.getLike());
    LOGGER.info("更新文章阅读数#articleId：{},dto：{}", dto.getArticleId(),
JSON.toJSONString(dto),rows);
    return ResponseResult.okResult(rows);
}
```

（3）监听类

com.heima.article.kafka.listener.ArticleIncrHandleListener

```
/**
 * 增量文章状态处理
 */
@Component
public class ArticleIncrHandleListener implements
com.heima.kafka.KafkaListener<String,String> {
    static Logger logger  =
LoggerFactory.getLogger(ArticleIncrHandleListener.class);

    @Autowired
    KafkaTopicConfig kafkaTopicConfig;
    @Autowired
    ObjectMapper mapper;
    @Autowired
    AppArticleService appArticleService;

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

    @Override
    public void onMessage(ConsumerRecord<String, String> data, Consumer<?, ?>
consumer) {
        logger.info("receive Article Incr Handle message:{}",data);
        String value = (String)data.value();
        try {
            ArticleVisitStreamMessage message = mapper.readValue(value,
ArticleVisitStreamMessage.class);
            ArticleVisitStreamDto dto = message.getData();
            appArticleService.updateArticleView(dto);
        }catch (Exception e){
```

```
            logger.error("kafka send message[class:{}] to Article Incr Handle
failed:{}","ArticleIncrHandle.class",e);
        }
    }
}
```

3.2.6 综合测试

1，启动article微服务

2，在behavior微服务中添加一个单元测试，为一条有效的数据添加点赞或阅读数

```
@Autowired
private AppLikesBehaviorService appLikesBehaviorService;

@Test
public void testLikesSave(){
    ApUser user = new ApUser();
    user.setId(1l);
    AppThreadLocalUtils.setUser(user);
    LikesBehaviorDto dto = new LikesBehaviorDto();
    dto.setEntryId(10120);
    dto.setOperation((short)0);
    dto.setType((short)0);
    appLikesBehaviorService.saveLikesBehavior(dto);
}
```