文章发布&粉丝管理

- 熟悉FastDfs的封装集成
- 熟悉自媒体系统的核心功能需求
- 掌握VUE+Echarts的集成使用
- 掌握后台功能的通用封装技巧
- 熟悉跨平台富文本的处理方案

1 需求分析

1.1 功能需求

在自媒体后台中主要包含的功能有内容管理：素材管理、文章发布、内容列表的查看、评论列表查看、图文数据统计；粉丝管理：粉丝概况、粉丝画像、粉丝列表

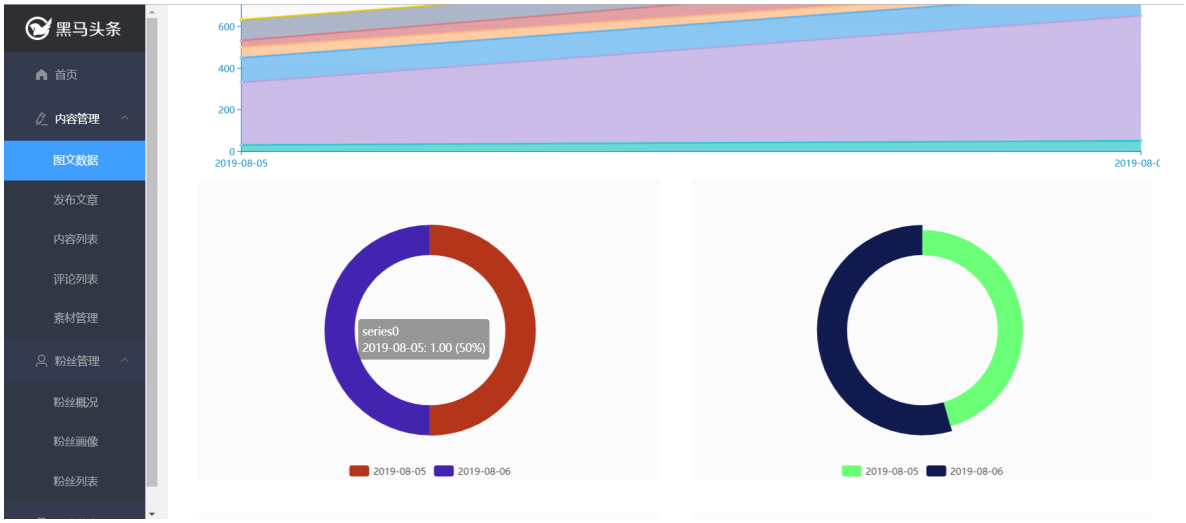
本案例开发功能包括：

- 根据不同时间范围查询图文明细数据
- 发布文章、保存草稿
- 根据状态查询当前用户的内容数据、修改、删除
- 素材查看、收藏素材、删除素材、取消收藏
- 粉丝概况：性别分布、年龄分布、终端分布、七日阅读量分布

1.2 前端需求

1.2.1 图文数据需求

在图文数据中我们主要实现自媒体用户所发布的文章的相关数据（发布量、阅读量、点赞量、评论量、收藏量、转发量、不喜欢）统计，为自媒体用户提供直观的运营数据。

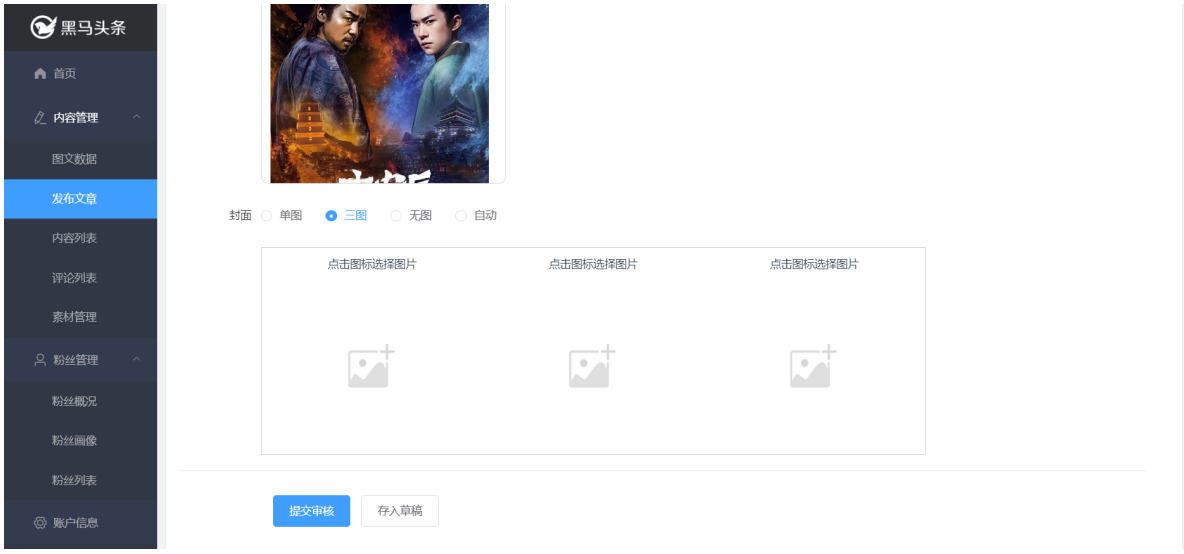


1.2.2 素材管理需求

素材管理部分的相关需求主要是实现图片的上传、删除、查询等功能，提供给自媒体人图文素材管理的空间。

1.2.3 发布文章需求

发布文章部分主要实现自媒体用户编辑文章内容，进行文章的发布或则保存草稿。

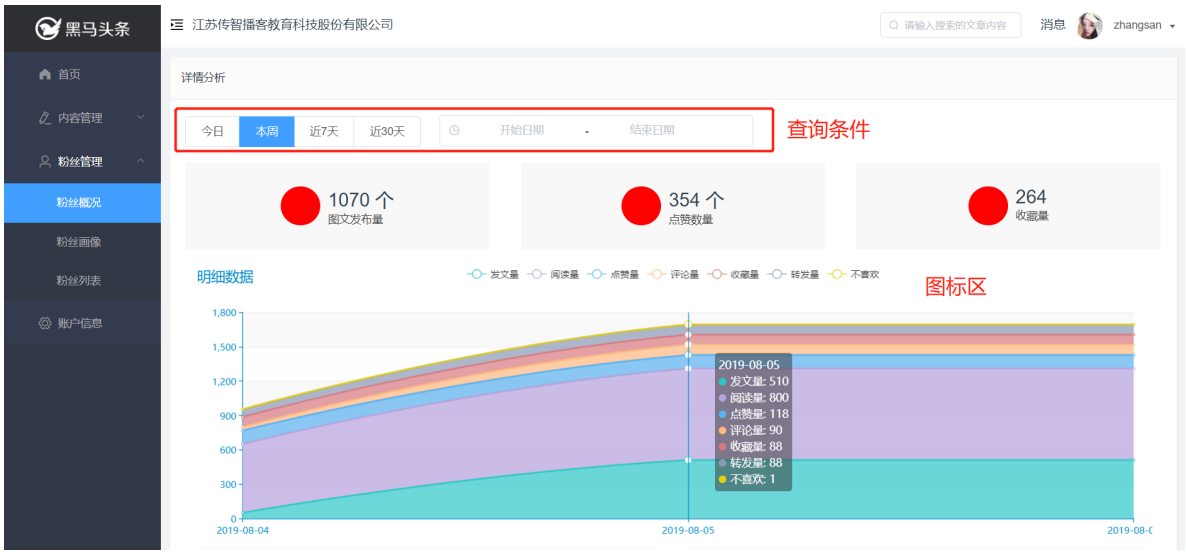


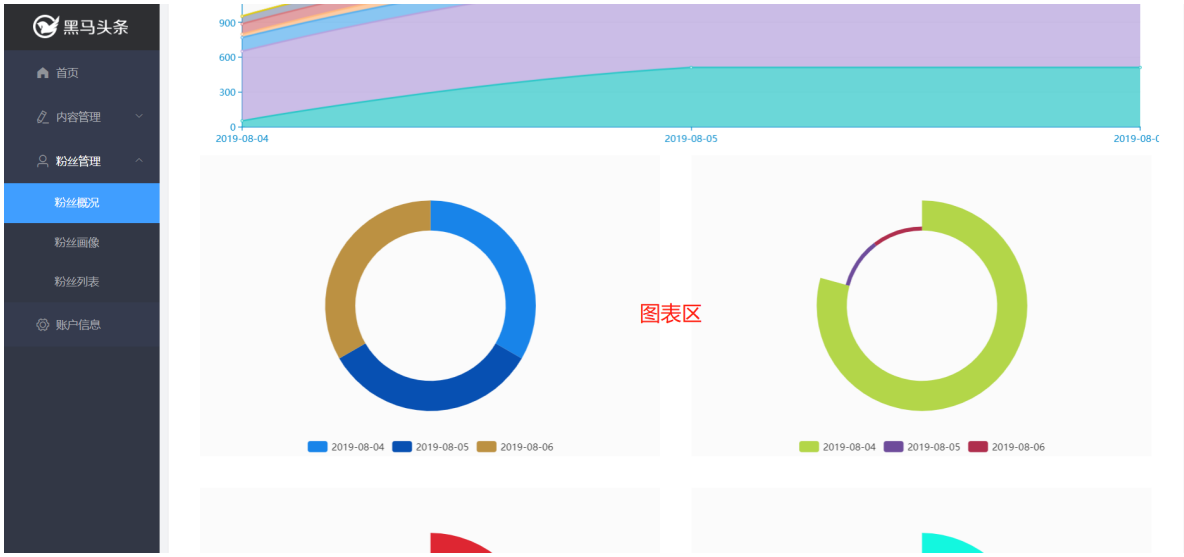
1.2.4 内容列表需求

内容列表功能模块， 主要实现根据不同的条件对文章内容进行查询，以及对文章的删除、修改等操作。

1.2.5 粉丝概况需求

粉丝概况和图文数据功能类似，但此处统计的数据只是当前自媒体用户的粉丝产生的相关数据。





2 定义

2.1 后端定义

2.1.1 路由定义

本功能会涉及以下相关数据表，用于读取文章内容和配置，存储在文章详情页面产生的各种行为，相关表的Mycat路由定义如下：

表名	描述	主键方式	存放DN	分表字段
wm_news_statistics	自媒体图文数据统计表	zk_sequence	DN[0~5]	burst=(id,user_id)
wm_sub_user	自媒体子账号信息表	auto_increment	DN[0~2]	parent_id
wm_user_auth	自媒体子账号权限信息表	auto_increment	DN[1~3]	user_id
wm_user	自媒体用户信息表	auto_increment	DN[0~5]	id
wm_user_login	自媒体用户登录行为信息表	auto_increment	DN[4]	user_id
wm_user_equipment	自媒体用户设备信息表	auto_increment	DN[1~3]	user_id
wm_fans_statistics	自媒体粉丝数据统计表	zk_sequence	DN[0~5]	burst=(id,user_id)
wm_fans_portrait	自媒体粉丝画像信息表	zk_sequence	DN[0~5]	burst=(id,user_id)

表名	描述	主键方式	存放DN	分表字段
wm_news	自媒体图文内容信息表	auto_increment	DN[0]	id
wm_material	自媒体图文素材信息表	auto_increment	DN[0]	id
wm_news_statistics	自媒体图文数据统计表	auto_increment	DN[0]	id

2.1.2 工程定义

自媒体服务：heima-leadnews-media

2.1.3 接口定义

自媒体后端，主要接口如下：

登录功能

- 登录

素材管理相关：

- 上传图片接口：用于上传用户提交的图片到素材库
- 删除图片接口：用于用户删除拥有的素材
- 收藏图片接口：收藏某个素材
- 取消收藏：取消收藏
- 素材列表：用户所有的素材

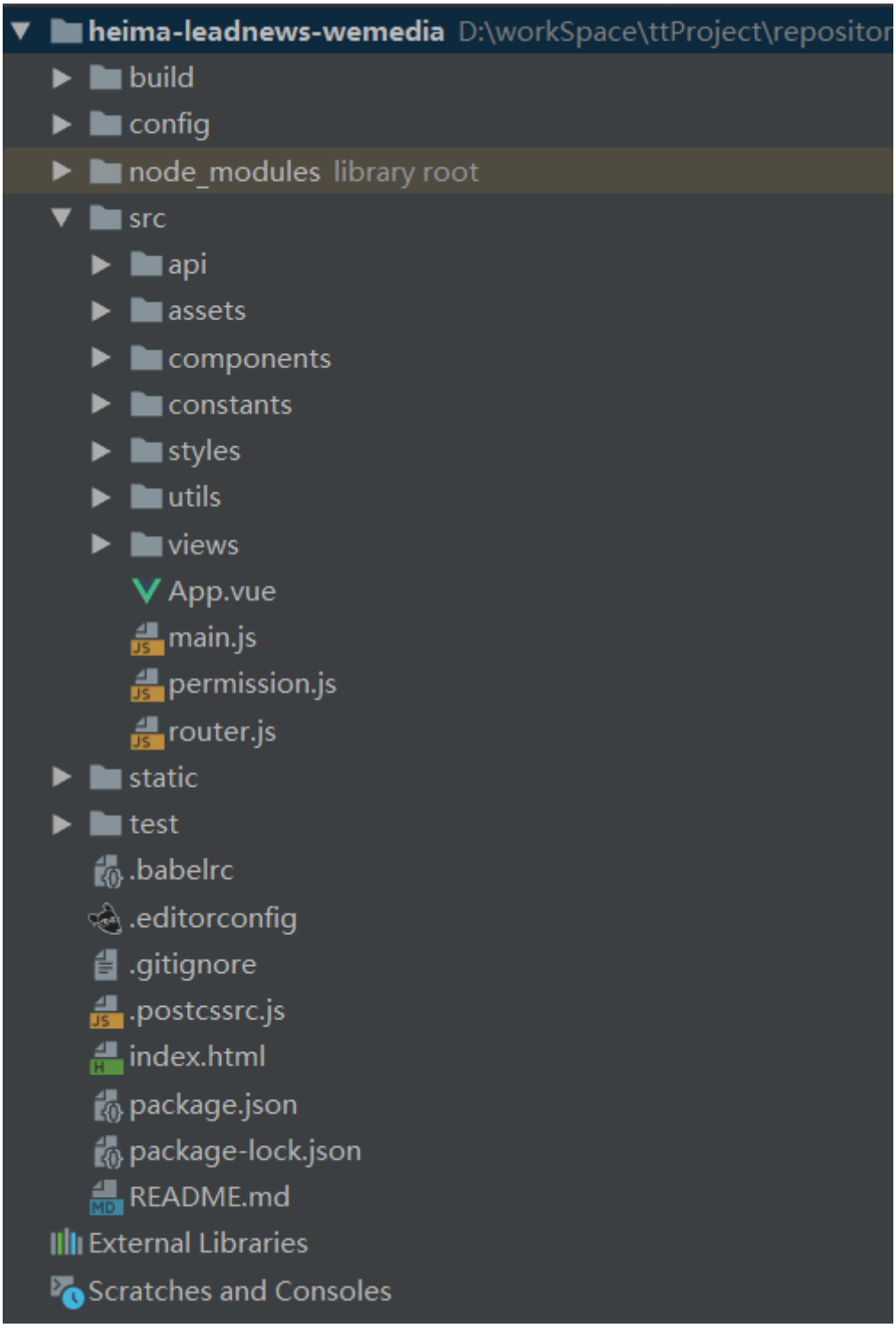
文章发布相关：

- 提交文章接口：用于提交文章
- 保存草稿接口：用于提交草稿文章
- 列表接口：用于查询当前用户的文章
- 详情接口接口：用于查询某篇文章详情
- 删除接口：用于删除谋篇文章

统计相关：

- 图文数据统计：当前自媒体用户文章被游客以及粉丝操作的相关数据
- 粉丝相关文章数据统计： 由粉丝对文章的相关操作产生的数据

2.1.4 结构定义



3 wemedia登录功能实现

3.1工程创建

在父工程下heima-leadnews创建模块heima-leadnews-media，从其他模块分别拷贝maven_dev.properties、 maven_prod.properties、maven_test.properties到项目的根目录

拷贝application.properties、log4j2.xml到项目的resources目录下，修改application.properties

```
server.port=${port.media}
spring.application.name=${sn.media}
```

从其他微服务下拷贝pom文件，创建对应模块的包名及引导类

分别引入mysql、jackson，security的配置

3.2登录功能后台

3.2.1 接口定义

参考标准	请参考通用接口规范
接口名称	/login/in
请求DTO	com.heima.model.media.pojos.WmUser
响应DTO	返回map{token:xxx,user:{...}}

3.2.2 mapper文件

WmUser 用户实体com.heima.model.media.pojos.WmUser

```
@Data
public class WmUser {
    private Long id;
    private String name;
    private String password;
    private String salt;
    private String nickname;
    private String image;
    private String location;
    private String phone;
    private Integer status;
    private String email;
    private Integer type;
    private Integer score;
    private Long apUserId;
    private Integer apAuthorId;
    private Date loginTime;
    private Date createdTime;
}
```

创建mapper接口：com.heima.model.mappers.wemedia.WmUserMapper

```
public interface WmUserMapper {
    WmUser selectByName(String name);
}
```

WmUserMapper.xml

```
<resultMap id="BaseResultMap" type="com.heima.model.media.pojos.WmUser" >
    <id column="id" property="id" />
    <result column="name" property="name"/>
    <result column="password" property="password"/>
    <result column="salt" property="salt"/>
    <result column="ap_user_id" property="apUserId"/>
    <result column="ap_author_id" property="apAuthorId"/>
```

```
    <result column="nickname" property="nickname"/>
    <result column="image" property="image"/>
    <result column="location" property="location"/>
    <result column="phone" property="phone"/>
    <result column="status" property="status"/>
    <result column="email" property="email"/>
    <result column="type" property="type"/>
    <result column="score" property="score"/>
    <result column="login_time" property="loginTime"/>
    <result column="created_time" property="createdTime"/>
</resultMap>
<sql id="Base_Column_List" >
    id, name, password, ap_user_id, ap_author_id, salt, nickname, image,
location, phone, status, email, type,
    score, login_time, created_time
</sql>
<!-- 通过名称查询用户 -->
<select id="selectByName" resultMap="BaseResultMap">
    select <include refid="Base_Column_List" />
    from wm_user where name = #{name} limit 1
</select>
```

3.2.3 代码实现

(1)创建接口com.heima.media.service.UserLoginService

```
public interface UserLoginService {
    ResponseResult login(WmUser user);
}
```

(2)实现类UserLoginServiceImpl

```
@Service
public class UserLoginServiceImpl implements UserLoginService {

    @Autowired
    private WmUserMapper wmUserMapper;

    public ResponseResult login(WmUser user){
        if
(StringUtils.isEmpty(user.getName())&&StringUtils.isEmpty(user.getPassword())) {
            return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_REQUIRE,"用户
名和密码不能为空");
        }
        WmUser wmUser = wmUserMapper.selectByName(user.getName());
        if(wmUser!=null){
            if(user.getPassword().equals(wmUser.getPassword())){
                Map<String,Object> map = Maps.newHashMap();
                wmUser.setPassword("");
                wmUser.setSalt("");
                map.put("token",AppJwtUtil.getToken(wmUser));
                map.put("user",wmUser);
                return ResponseResult.okResult(map);
            }else{
                return
ResponseResult.errorResult(AppHttpCodeEnum.LOGIN_PASSWORD_ERROR);
```

```
            }
        }else{
            return ResponseResult.errorResult(AppHttpCodeEnum.DATA_NOT_EXIST,"用户不存在");
        }
    }
}
```

(3)创建apis接口

```
public interface LoginControllerApi {
    public ResponseResult login(WmUser user);
}
```

（4）实现controller:com.heima.media.controller.v1.LoginController

```
@RestController
@RequestMapping("/login")
public class LoginController implements LoginControllerApi {

    @Autowired
    private UserLoginService userLoginService ;

    @Override
    @RequestMapping("/in")
    public ResponseResult login(@RequestBody WmUser user){
        return userLoginService.login(user);
    }
}
```

3.3 登录功能前台

3.3.1 定义api

在src/constants/api.js中定义常量映射到后端请求地址

```
export const  API_USERAUTH = '/login/in' //用户认证
```

在src/api/login.js中定义请求方法,在请求成功之后，需要把后台返回的token数据写入本地缓存

```
import request from '@/utils/request'
import {setUser} from '@/utils/store'
import  { API_GETPHONECODE , API_USERAUTH , API_CAPTCHAS }  from '@/constants/api'

export function loginByUsername(name,password) {
  const data = {
    name,password
  }
  return request({
    url: API_USERAUTH,
    method: 'post',
```

```
    data
  }).then(result => {
      if(result['code']==0){
        let temp = result.data
         setUser({name:temp.user.name,photo:null,token:temp.token}) //设置用户的个
人数据
      }
       return result
  })
}
```

3.3.2 路由调整

在src/router.js中asyncRouterMap对象的children数组中增加以下改动，以满足全局自动记录路由的功能：

设置登录为起始路由

```
export const asyncRouterMap = [
  {
    path: "/",
    component: Layout,
    redirect:'/login', //默认子路由
    name:'mainIndex',
    children:[
      {
        path:'/index',
        component: () => import('@/views/dashboard/index.vue'),
      }
    ]
  },
  {
    path: '/login',
    component: () => import('@/views/login/index.vue'),
  },
  {
    path: '*',
    component: () => import('@/views/404.vue'),
  }
]
var  myRouter = new Router({
  routes: asyncRouterMap
})
export default myRouter
```

3.3.3 实现登录功能

```
<template>
  <div class="login">
    <div class="container">
      <img src="@/assets/logo_index.png" class="user-avatar">
      <el-form :model="ruleForm" status-icon :rules="rules" ref="ruleForm"
class="login-ruleForm">
        <el-form-item prop="name">
          <el-input type="text" v-model="ruleForm.name" autocomplete="off"
placeholder="请输入账户名"></el-input>
```

```
        </el-form-item>
        <el-form-item prop="password">
          <el-input type="password" v-model="ruleForm.password"
autocomplete="off" placeholder="请输入密码"></el-input>
        </el-form-item>
        <div class="allow">
          <div id="myCode"></div>
          <el-checkbox v-model="checked"></el-checkbox>我已阅读并同意<a href="">用
户协议</a>和<a href="">隐私条款</a>
        </div>
        <el-form-item class="loginBtn">
          <el-button type="primary" @click="submitForm('ruleForm')">登录</el-
button>
        </el-form-item>
      </el-form>
    </div>
  </div>
</template>

<script>
import gt from  '@/components/gt' //人机交互验证码
import { loginByUsername , getMobileCode , getCaptchas } from '@/api/login'
export default {
  data() {
    var validateName = (rule, value, callback) => {
      if (value === '') {
        callback(new Error('请输入登录用户名'));
      } else {
        callback();
      }
    };
    var validatePass = (rule, value, callback) => {
      if (value === '') {
        callback(new Error('请输入密码'));
      } else {
        callback();
      }
    };
    return {
      checked: true,
      ruleForm: {
        name: '',
        password: '',
      },
      rules: {
        name: [
          { validator: validateName, trigger: 'blur' }
        ],
        password: [
          { validator: validatePass, trigger: 'blur' }
        ],
      }
    };
  },
  components: {
  },
  computed: {
```

```
  },
  methods: {
  async submitForm () {
        let {password , name} = this.ruleForm;
        if(!name || !password){
            this.$message({
               message:'用户名和密码不能为空',
               type:'warning'
             })
             return
        }
        //登录
        let result = await loginByUsername(name,password) //登录
         if(result.code==0){
           this.$router.replace({path:'/index'}) //跳转
         }else{
           this.$message({
             message:result.errorMessage,
             type:'error'
           })
         }
    }
  }
}
</script>
<style rel="stylesheet/scss" lang="scss" scoped>
.login {
  background-image: url('../../assets/login_bg.jpg');
  background-size: cover;
  height: 100vh;
  display: flex;
  justify-content: center;
  align-items: center;
  .container {
    background-color: #ffffff;
    width: 30%;
    padding: 30px 0;
    img {
      width: 40%;
    }
  }
  .login-ruleForm {
    padding: 25px 40px 0;
    .allow {
      text-align: left;
      font-size: 14px;
      margin-bottom: 20px;
      color: #999999;
      a {
        color: #3296fa;
      }
      .el-checkbox {
        margin-right: 10px;
      }
    }
    .el-form-item {
      margin-bottom: 20px;
    }
```

```
    .checkCode {
      .el-input {
        width: 60%;
        float: left;
      }
      .el-button {
        width: 35%;
        float: right;
        span{
          width: 100%;
          display: inline-block;
        }
      }
    }
    .loginBtn {
      .el-button {
        width: 100%;
      }
    }
  }
}
</style>
```

4 素材管理开发

4.1 FASTDFS封装

4.1.1 FASTDFS配置

- 在项目根目录pom.xml中增加以下配置

```
<properties><fastdfs.version>0.2.0</fastdfs.version></properties>
<!-- fastdfs客服端 -->
<dependency>
    <groupId>com.luhuiguo</groupId>
    <artifactId>fastdfs-spring-boot-starter</artifactId>
    <version>${fastdfs.version}</version>
    <exclusions>
        <exclusion>
            <artifactId>logback-classic</artifactId>
            <groupId>ch.qos.logback</groupId>
        </exclusion>
    </exclusions>
</dependency>
```

- 在common项目pom.xml中增加以下配置

```
<!--fastdfs-->
<dependency>
    <groupId>com.luhuiguo</groupId>
    <artifactId>fastdfs-spring-boot-starter</artifactId>
    <exclusions>
        <exclusion>
            <artifactId>logback-classic</artifactId>
            <groupId>ch.qos.logback</groupId>
        </exclusion>
    </exclusions>
</dependency>
```

- 在common\src\main\resources下创建文件fast-dfs.properties

```
fast.dfs.connect-timeout=3000
fast.dfs.so-timeout=6000
fast.dfs.tracker-server=192.168.25.133:22122
```

4.1.2 FastDfsConfig

在common下创建类：com.heima.common.fastdfs.FastDfsConfig，重载自动装载dfs和设定连接池。

```
/**
 * 自动化配置核心数据库的连接配置
 */
@Setter
@Getter
@Configuration
@ConfigurationProperties(prefix="fast.dfs")
@PropertySource("classpath:fast-dfs.properties")
public class FastDfsConfig extends  FdfsAutoConfiguration {

    int soTimeout;
    int connectTimeout;
    String trackerServer;

    public FastDfsConfig(FdfsProperties properties){
        super(properties);
    }

    @Bean
    public PooledConnectionFactory pooledConnectionFactory() {
        PooledConnectionFactory pooledConnectionFactory = new
PooledConnectionFactory();
        pooledConnectionFactory.setSoTimeout(getSoTimeout());
        pooledConnectionFactory.setConnectTimeout(getConnectTimeout());
        return pooledConnectionFactory;
    }


    @Bean
    public TrackerConnectionManager trackerConnectionManager(FdfsConnectionPool
fdfsConnectionPool) {
```

```
        return new TrackerConnectionManager(fdfsConnectionPool,
Arrays.asList(trackerServer));
    }
}
```

4.1.3 FastDfsClient

在common下创建类com.heima.fastdfs.FastDfsClient，封装dfs上传、下载等常用方法：

```
/**
 * dfs客服端
 */
@Component
public class FastDfsClient {

    @Autowired
    FastFileStorageClient storageClient;

    /**
     * 上传文件方法
     * <p>Title: uploadFile</p>
     * <p>Description: </p>
     * @param fileName 文件全路径
     * @param extName 文件扩展名，不包含（.）
     * @return
     * @throws Exception
     */
    public String uploadFile(String fileName, String extName) throws Exception {
        StorePath s = storageClient.uploadFile(FileUtils.readFileToByteArray(new
File(fileName)),extName);
        String result = s.getFullPath();
        return result;
    }

    public String uploadFile(String fileName) throws Exception {
        return uploadFile(fileName, null);
    }

    /**
     * 上传文件方法
     * <p>Title: uploadFile</p>
     * <p>Description: </p>
     * @param fileContent 文件的内容，字节数组
     * @param extName 文件扩展名
     * @return
     * @throws Exception
     */
    public String uploadFile(byte[] fileContent, String extName) throws
Exception {
        StorePath s = storageClient.uploadFile(fileContent,extName);
        String result = s.getFullPath();
        return result;
    }

    public String uploadFile(byte[] fileContent) throws Exception {
        return uploadFile(fileContent, null);
    }
```

```
    /**
     * 文件下载方法
     */
    public byte[] downFile(String fileId) throws Exception {
        return storageClient.downloadFile("",fileId);
    }

    /**
     * 文件下载方法
     */
    public byte[] downGroupFile(String group, String fileId) throws Exception {
        return storageClient.downloadFile(group,fileId);
    }

    public int delFile(String fileId) throws Exception {
         storageClient.deleteFile(fileId);
         return 1;
    }

}
```

4.2 素材上传接口

4.2.1

4.2.2 接口定义

（1）基本定义

参考标准	请参考通用接口规范
接口名称	/api/v1/media/material/upload_picture
请求DTO	MultipartFile
响应DTO	WmMaterial

（2）CODE定义

PARAM_INVALID	PARAM_INVALID(501,"无效参数"),
PARAM_IMAGE_FORMAT_ERROR	PARAM_IMAGE_FORMAT_ERROR(502,"图片格式有误")
SERVER_ERROR	SERVER_ERROR(503,"服务器内部错误"),

4.2.3 类定义

类说明：

- MultipartFile是用于接收用户上传文件
- FastDFSClient用于将用户上传的图片上传至图片服务器，放置在common模块
- WmMaterialMapper、WmNewsMaterialMapper是MybatisMapper文件，放置在model模块
- MaterialManageControllerApi是服务接口定义，放置在apis模块

- MaterialManageController、MaterialService、MaterialServiceImpl是对功能的实现，放置在media模块

4.2.4 Mapper实现

（1）WmMaterial 实体

在model模块下创建类com.heima.model.media.pojos.WmMaterial

```
@Data
public class WmMaterial {
    private Integer id;
    @IdEncrypt
    private Long userId;
    private String url;
    private short type;
    private Short isCollection;
    private Date createdTime;
}
```

（2）WmMaterialMapper

创建类com.heima.model.mappers.wemedia.WmMaterialMapper，增加素材插入方法：

```
public interface WmMaterialMapper {
    int insert(WmMaterial record);
}
```

（3）WmMaterialMapper.xml

同样在model模块中创建文件resources/mappers/wemedia/WmMaterialMapper.xml，并写出接口对应sql， 保存上传图片的信息到数据库中。

```
<mapper namespace="com.heima.model.mappers.wemedia.WmMaterialMapper">
    <insert id="insert" parameterType="com.heima.model.media.pojos.WmMaterial"
useGeneratedKeys="true" keyProperty="id">
    insert into wm_material (user_id, url,
    type, is_collection, created_time
    )
    values (#{userId}, #{url},
    #{type}, #{isCollection}, #{createdTime}
    )
    </insert>
</mapper>
```

4.2.5 service思路分析

- 判断入参multipartFile是否合法，不合法则返回PARAM_INVALID错误
- 判断入参multipartFile是否有合法的扩展名，不合法则返回PARAM_INVALID错误
- 上传图片到FastDFS服务器
- 上传图片到服务器失败
- 上传图片流程完成， 返回信息给前端

4.2.6 代码实现

（1）MaterialService

在media模块中（若模块不存在则创建模块儿）创建类：com.heima.media.service.MaterialService，并添加uploadPicture方法实现图片的上传逻辑

定义获取文章详情接口：

```
public interface MaterialService {
    /**
    * 上传图片接口*
    * @param multipartFile*
    * @return*
    */
    ResponseResult uploadPicture(MultipartFile multipartFile);
}
```

（2）MaterialServiceImpl

同样在media中创建类：com.heima.media.service.impl.MaterialServiceImpl，在方法的实现中首先调用fastDFS实现图片上传至服务器， 然后将文件信息存储到关系型数据库中。

修改工程resources/application.properties文件，添加配置

```
FILE_SERVER_URL=http://192.168.25.133/
```

实现类

```
@Service
@Slf4j
public class MaterialServiceImpl implements MaterialService {

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

    @Autowired
    private FastDfsClient fastDFSClient;

    @Autowired
    private WmMaterialMapper wmMaterialMapper;

    @Override
    public ResponseResult uploadPicture(MultipartFile multipartFile) {
        WmUser user = WmThreadLocalUtils.getUser();
        if (multipartFile == null) {
            return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_INVALID);
        }
        String originFileName = multipartFile.getOriginalFilename();
        String extName =
originFileName.substring(originFileName.lastIndexOf(".") +
        1);
        if(!extName.matches("(gif|png|jpg|jpeg)")) {
            return

 ResponseResult.errorResult(AppHttpCodeEnum.PARAM_IMAGE_FORMAT_ERROR);
        }
        // StringBuilder imgUrl = new StringBuilder(fileServerUrl);
        String fileId = null;
```

```
        //上传图片获得文件id
        try {
            fileId = fastDFSClient.uploadFile(multipartFile.getBytes(),
extName);
        } catch (Exception e) {
            e.printStackTrace();
            log.error("user {} upload file {} to fastDFS error, error info:n",
            user.getId(),
            originFileName, e.getMessage());
            return ResponseResult.errorResult(AppHttpCodeEnum.SERVER_ERROR);
        }
        //上传成功保存媒体资源到数据库
        WmMaterial wmMaterial = new WmMaterial();
        wmMaterial.setCreatedTime(new Date());
        wmMaterial.setType((short) 0);
        wmMaterial.setUrl(fileId);
        wmMaterial.setUserId(user.getId());
        wmMaterial.setIsCollection((short) 0);
        wmMaterialMapper.insert(wmMaterial);
        //设置返回值
        wmMaterial.setUrl(fileServerUrl + wmMaterial.getUrl());
        return ResponseResult.okResult(wmMaterial);
    }
}
```

（3）MaterialManageControllerApi

创建类：com.heima.media.apis.MaterialManageControllerApi，在此类中定义控制器接口。

此类在apis模块中创建，定义了相关接口，实现如下：

```
public interface MaterialManageControllerApi {
    /**
    * 上传图片
    * @param multipartFile
    * @return
    */
    ResponseResult uploadPicture(MultipartFile multipartFile);
}
```

（4）MaterialManageController

创建类：com.heima.media.controller.v1.MaterialManageController， 在控制器中调用Service中写的方法即可

该类的实现较为简单，引入Service并调用即可：

```
@RestController
@RequestMapping("/api/v1/media/material")
public class MaterialManageController implements MaterialManageControllerApi{

    @Autowired
    private MaterialService materialService;

    @PostMapping("/upload_picture")
    @Override
    public ResponseResult uploadPicture(MultipartFile file) {
        return materialService.uploadPicture(file);
    }
}
```

4.3 删除图片接口

4.3.1 接口定义

（1）基本定义

此接口用于删除无关联的图片。

参考标准	请参考通用接口规范
接口名称	/api/v1/media/material/del_picture
请求DTO	com.heima.model.media.dtos.WmMaterialDto
响应DTO	{ "host": null, "code": 0, "error_message": "操作成功", "data": "SUCCESS" }

（2）CODE定义

PARAM_INVALID	PARAM_INVALID(501,"无效参数")
SERVER_ERROR	SERVER_ERROR(503,"服务器内部错误"),

4.3.2 类定义

类说明：

- 涉及的pojo和Mapper都存储在model模块中
- 请求DTO也重用WmMaterialDto，缺少字段进行补充即可
- Service、Controller等类都进行重用，定义新的方法

4.3.3 Mapper实现

相关类在model模块中实现，之后所有的mapper文件、dto实体类都默认在model模块儿中进行实现，service相关接口默认在media模块中实现。

（1）WmMaterialDto

创建类com.heima.model.media.dtos.WmMaterialDto，用于接收前端传递过来的参数。

```
@Data
public class WmMaterialDto {
    @IdEncrypt
    private Integer id;
}
```

（2）WmMaterialMapper

在com.heima.model.mappers.wemedia.WmMaterialMapper类中定义方法：

- selectByPrimaryKey，依据id查询媒体文件
- deleteByPrimaryKey，根据id删除图片

```
public interface WmMaterialMapper {
    WmMaterial selectByPrimaryKey(Integer id);
    int deleteByPrimaryKey(Integer id);
}
```

（3）WmMaterialMapper.xml

在文件resources/mappers/wemedia/WmMaterialMapper.xml，在当前文件中根据业务写出对应的SQL

```
<resultMap id="BaseResultMap" type="com.heima.model.media.pojos.WmMaterial" >
      <id column="id" />
      <result column="user_id" />
      <result column="url"/>
      <result column="type"/>
      <result column="is_collection"/>
      <result column="created_time" />
</resultMap>
<sql id="Base_Column_List" >
    id, user_id, url, type, is_collection, created_time
</sql>
<select id="selectByPrimaryKey"
resultType="com.heima.model.media.pojos.WmMaterial"
parameterType="java.lang.Integer" >
    select
    <include refid="Base_Column_List" />
    from wm_material
    where id = #{id}
</select>

<delete id="deleteByPrimaryKey" parameterType="java.lang.Integer" >
    delete from wm_material  where id = #{id}
</delete>
```

（4）WmNewsMaterialMapper

删除时，如果对应的material有关联引用，则不能删除，所以需要查询对应的引用数据：com.heima.model.mappers.wemedia.WmNewsMaterialMapper

```
public interface WmNewsMaterialMapper {
    int countByMid(Integer mid);
}
```

(2)WmNewsMaterialMapper.xml

在文件resources/ mappers/wemedia/WmNewsMaterialMapper.xml SQL如下：

```
<mapper namespace="com.heima.model.mappers.wemedia.WmNewsMaterialMapper">
<select id="countByMid" resultType="java.lang.Integer">
    select count(0)
    from wm_news_material
    where material_id = #{mid}
</select>

</mapper>
```

4.3.4 时序说明

- 判断行为实体参数是否存在，如果不存在则返回PARAM_REQUIRE错误
- 查看当前删除图片是否存在于系统中
- 当前图片是否被引用， 如果被引用则不可删除
- 删除图片服务器上面的图片
- 所有操作完成成功返回

4.3.5 代码实现

(1)MaterialService

在com.heima.media.service.MaterialService 中新增方法delPicture实现图片的删除逻辑，定义获取文章详情接口：

```
ResponseResult delPicture(WmMaterialDto dto);
```

(2)MaterialServiceImpl

在类com.heima.media.service.impl.MaterialServiceImpl中实现图片资源管理。此处主要实现了图片资源的删除，删除逻辑主要分为两步第一步先删除fastDFS上面的文件，然后删除数据库中的关联关系。

```
@Override
public ResponseResult delPicture(WmMaterialDto dto) {
    WmUser user = WmThreadLocalUtils.getUser();
    if (dto == null || dto.getId() == null) {
        return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_INVALID);
    }
    //删除fastDFS上的文件
    WmMaterial wmMaterial = wmMaterialMapper.selectByPrimaryKey(dto.getId());
    if (wmMaterial == null) {
        return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_INVALID);
    }
    int count = wmNewsMaterialMapper.countByMid(dto.getId());
    if (count > 0) {
        return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_INVALID,
                                          "当前图片被引用");
    }
    String fileId = wmMaterial.getUrl().replace(fileServerUrl, "");
    try {
        fastDFSClient.delFile(fileId);
    } catch (Exception e) {
```

```
            e.printStackTrace();
            log.error("user {} delete file {} from fastDFS error, error info:n",
                      user.getId(),
                      fileId, e.getMessage());
            return ResponseResult.errorResult(AppHttpCodeEnum.SERVER_ERROR);
        }
        //删除数据库记录
        wmMaterialMapper.deleteByPrimaryKey(dto.getId());
        return ResponseResult.okResult(AppHttpCodeEnum.SUCCESS);
    }
```

(3)MaterialManageControllerApi

在类com.heima.media.apis.MaterialManageControllerApi中定义图片删除接口方法。

```
ResponseResult delPicture(WmMaterialDto wmMaterial);
```

(4)MaterialManageController

在类com.heima.media.controller.v1.MaterialManageController中实现图片删除接口方法。

```
@PostMapping("/del_picture")
@Override
public ResponseResult delPicture(@RequestBody WmMaterialDto dto) {
    return materialService.delPicture(dto);
}
```

4.4 素材列表接口

4.4.1 接口定义

（1）基本定义

此接口用于加载自媒体人的图文素材，和用于图片素材选择框。

参考标准	请参考通用接口规范
接口名称	/api/v1/media/material/list
请求DTO	com.heima.model.media.dtos.WmMaterialListDto
响应DTO	{"host": null,"code": 0, "error_message": "操作成功", "data":{size:1,total:1,list:[]}}

（2）CODE定义

PARAM_INVALID	PARAM_INVALID(501,"无效参数")
SERVER_ERROR	SERVER_ERROR(503,"服务器内部错误"),

4.4.2 类定义

类说明：

- 涉及的pojo和Mapper都存储在model模块中
- 请求DTO也重用WmMaterialDto，缺少字段进行补充即可

- Service、Controller等类都进行重用，定义新的方法

4.4.3 Mapper实现

相关类在model模块中实现

(1)PageRequestDto

创建类com.heima.model.common.dtos.PageRequestDto，再该类中定义了checkParam方法用于校验分页参数， 若分页参数异常并给分页参数默认值。

```
@Data
@Slf4j
public class PageRequestDto {

    protected Integer size;
    protected Integer page;

    public void checkParam() {
        if (this.page == null || this.page < 0) {
            setPage(1);
        }
        if (this.size == null || this.size < 0 || this.size > 100) {
            setSize(10);
        }
    }
}
```

(2)WmMaterialListDto

创建类com.heima.model.media.dtos.WmMaterialListDto，该类继承了PageRequestDto用于实现分页参数的封装

```
@Data
public class WmMaterialListDto extends PageRequestDto {
    Short isCollected; //1 查询收藏的
}
```

(3)WmMaterialMapper

在类com.heima.model.mappers.wemedia.WmMaterialMapper新增接口方法
findListByUidAndStatus用于根据用户id和需要查询的图片状态（是否收藏）查询图片、
countListByUidAndStatus进行分页统计：

```
List<WmMaterial> findListByUidAndStatus(WmMaterialListDto dto, Long uid);
int countListByUidAndStatus(WmMaterialListDto dto, Long uid);
```

(4)WmMaterialMapper.xml

在文件resources/mappers/wemedia/WmMaterialMapper.xml中新增以下内容，对应在接口中新增的方法。

```
<select id="findListByUidAndStatus"
        resultType="com.heima.model.media.pojos.WmMaterial">
    select
    <include refid="Base_Column_List" />
```

```
    from wm_material
    where user_id = #{uid}
    <if test="dto.isCollected == 1">
        and is_collection = #{dto.isCollected}
    </if>
    limit ${(dto.page - 1) * dto.size}, ${dto.size}
</select>
<select id="countListByUidAndStatus" resultType="java.lang.Integer">
    select count(0) from wm_material where user_id = #{uid}
    <if test="dto.isCollected == 1">
        and is_collection = #{dto.isCollected}
    </if>
</select>
```

4.4.4 时序说明

- 检测分页参数是否正确，如果不存在则返回PARAM_REQUIRE错误
- 操作成功，返回查询结果集

4.4.5 代码实现

(1)MaterialService

在com.heima.media.service.MaterialService中新增方法findList查找图片列表的接口方法：

```
ResponseResult findList(WmMaterialListDto dto);
```

(2)MaterialServiceImpl

在com.heima.media.service.impl.MaterialServiceImpl，中实现findList方法用于实现查找图片列表

```
@Override
public ResponseResult findList(WmMaterialListDto dto) {
    dto.checkParam();
    Long uid = WmThreadLocalUtils.getUser().getId();
    List<WmMaterial> datas = wmMaterialMapper.findListByUidAndStatus(dto,
                                                                     uid);
    datas = datas.stream().map((item) -> {
        item.setUrl(fileServerUrl + item.getUrl());
        return item;
    }).collect(Collectors.toList());
    int total = wmMaterialMapper.countListByUidAndStatus(dto, uid);
    Map<String, Object> resDatas = new HashMap<>();
    resDatas.put("curPage", dto.getPage());
    resDatas.put("size", dto.getSize());
    resDatas.put("list", datas);
    resDatas.put("total", total);
    return ResponseResult.okResult(resDatas);
}
```

（3）MaterialManageControllerApi

在类com.heima.media.apis.MaterialManageControllerApi中定义了相关接口，实现如下：

```
ResponseResult list(WmMaterialListDto dto);
```

（4）MaterialManageController

在类com.heima.media.controller.v1.MaterialManageController中增加对应接口方法。

```
@RequestMapping("/list")
@Override
public ResponseResult list(@RequestBody WmMaterialListDto dto) {
    return materialService.findList(dto);
}
```

4.5 收藏、取消收藏接口

4.5.1 接口定义

（1）基本定义

此接口用于标记素材图片收藏、取消收藏等操作。

参考标准	请参考通用接口规范
收藏接口名称	/api/v1/media/material/collect
取消收藏接口	/api/v1/media/material/cancle_collect
请求DTO	com.heima.model.media.dtos.WmMaterialDto
响应DTO	{"host": null, "code": 0,"error_message": "操作成功","data": "SUCCESS"}

（2）CODE定义

PARAM_INVALID	PARAM_INVALID(501,"无效参数")
SERVER_ERROR	SERVER_ERROR(503,"服务器内部错误"),

4.5.2 类定义

类说明：

- 涉及的pojo和Mapper都存储在model模块中
- 请求DTO也重用WmMaterialDto，缺少字段进行补充即可
- Service、Controller等类都进行重用，定义新的方法

4.5.3 Mapper实现

相关类在model模块中实现

（1）WmMaterialDto

创建类com.heima.model.media.dtos.WmMaterialDto，该类不会输出给前端，所以相关属性可不做混淆加密设置。

```
@Data
public class WmMaterialDto {
    @IdEncrypt
    private Integer id;
}
```

(2)WmMaterialMapper

在类com.heima.model.mappers.wemedia.WmMaterialMapper中定义按照文章ID查询内容方法：

```
int updateStatusByUidAndId(Integer id, Long userId, Short type);
```

(3)WmMaterialMapper.xml

在WmMaterialMapper.xml文件中增加以下配置

```
<update id="updateStatusByUidAndId">
    update wm_material
    set is_collection = #{type}
    where user_id = #{userId} and id = #{id}
</update>
```

4.5.4 时序说明

(1)收藏时序图

- 判断参数是否符合要求，如果不符合在则返回PARAM_REQUIRE错误
- 更新当前素材的状态
- 所有操作完成成功返回

(2)取消收藏时序图

- 判断参数是否符合要求，如果不符合在则返回PARAM_REQUIRE错误
- 更新当前素材的状态
- 所有操作完成成功返回

4.5.5 代码实现

(1)MaterialService

在类com.heima.media.service.MaterialService中定义修改素材收藏状态的接口：

```
ResponseResult changeUserMaterialStatus(WmMaterialDto dto, Short type);
```

(2)MaterialServiceImpl

在类com.heima.media.service.impl.MaterialServiceImpl中增加素材收藏或取消方法。

```
@Override
public ResponseResult changeUserMaterialStatus(WmMaterialDto dto, Short
    type) {
        if (dto == null || dto.getId() == null) {
            return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_INVALID);
        }
        WmUser user = WmThreadLocalUtils.getUser();
        wmMaterialMapper.updateStatusByUidAndId(dto.getId(), user.getId(),
type);
        return ResponseResult.okResult(AppHttpCodeEnum.SUCCESS);
    }
```

(3)MaterialManageControllerApi

在类com.heima.media.apis.MaterialManageControllerApi中定义了相关接口方法：

```
    ResponseResult collectionMaterial(WmMaterialDto dto);
    ResponseResult cancleCollectionMaterial(WmMaterialDto dto);
```

(4)MaterialManageController

定义常量：com.heima.common.media.constans.WmMediaConstans

```
public class WmMediaConstans {
    public static final Short COLLECT_MATERIAL= 1; //收藏
    public static final Short CANCEL_COLLECT_MATERIAL = 0; //取消收藏
    public static final String WM_NEWS_TYPE_IMAGE = "image";
    public static final Short WM_NEWS_DRAFT_STATUS = 0; //草稿
    public static final Short WM_NEWS_SUMMIT_STATUS = 1; //提交
    public static final Short WM_NEWS_AUTHED_STATUS = 8; //审核通过
    public static final Short WM_NEWS_PUBLISH_STATUS = 9; //已发布


    public static final Short WM_NEWS_NONE_IMAGE = 0; //无图
    public static final Short WM_NEWS_SINGLE_IMAGE = 1; //单图
    public static final Short WM_NEWS_MANY_IMAGE = 3; //多图
    public static final Short WM_NEWS_TYPE_AUTO = -1; //图文类型自动

    public static final Short WM_CONTENT_REFERENCE = 0;
    public static final Short WM_IMAGE_REFERENCE = 1;


    public static final char WM_NEWS_IMAGES_SWPARATOR = ',';
    public static final short WM_NEWS_STATISTIC_CUR = 0; //查询当日
    public static final short WM_NEWS_STATISTIC_WEEK = 1; //查询近一周
    public static final short WM_NEWS_STATISTIC_NEAR7 = 7; //查询近七天
    public static final short WM_NEWS_STATISTIC_NEWA30 = 30; //查询近30天

}
```

在类com.heima.media.controller.v1.MaterialManageController中增加对应接口方法。

```
@PostMapping("/collect")
@Override
public ResponseResult collectionMaterial(@RequestBody WmMaterialDto dto) {
    return materialService.changeUserMaterialStatus(dto,

 WmMediaConstans.COLLECT_MATERIAL);
}

@PostMapping("/cancel_collect")
@Override
public ResponseResult cancleCollectionMaterial(@RequestBody WmMaterialDto
    dto) {
        return materialService.changeUserMaterialStatus(dto,
        WmMediaConstans.CANCEL_COLLECT_MATERIAL);
    }
```

4.6 素材管理前台

导入资料文件夹中的heima-leadnews-wemedia项目，使用web strom打开

4.6.1 定义api

(1)图片列表

- 在src/constants/api.js中定义常量映射到后端请求地址

```
export const API_USERIMAGES_LIST = '/api/v1/media/material/list'
```

- 在src/api/publish.js中定义请求方法（此处省略了引入刚才定义的常量，此后所有导入省略请自行导入需要的常量及方法）

```
//拉取全部的素材图片
export function getAllImgData (data) {
    return Request({
    url:API_USERIMAGES_LIST,
    method:'post',
    params:{},
    data:data
    })
}
```

（2）删除图片

- 在src/constants/api.js中定义常量映射到后端请求地址

```
export const API_MODIFYIMAGE_DELETE ='/api/v1/media/material/del_picture' //删除图片
```

- 在src/api/publish.js中定义请求方法

```
//删除图片素材
export function delImg (id) {
    return Request({
        url:API_MODIFYIMAGE_DELETE,
        method:'post',
        params:{},
        data:{id:id}
    })
}
```

（3）收藏或取消收藏图片

- 在src/constants/api.js中定义常量映射到后端请求地址

```
export const API_MODIFYIMAGE_COL = '/api/v1/media/material/collect'
//收藏用户素材 或 修改收藏状态接
export const API_MODIFYIMAGE_COL_CANCEL =
'/api/v1/media/material/cancel_collect' //取消用户素材 或 修改收藏状态接口
```

- 在src/api/publish.js中定义请求方法

```
//收藏或取消收藏方法
export function collectOrCancel (id,data) {
    let collect = data.isCollected
```

```
    let url = API_MODIFYIMAGE_COL
    if(collect==0){
        url = API_MODIFYIMAGE_COL_CANCEL
    }
    return Request({
        url:url,
        method:'post',
        params:{},
        data:{id:id}
    })
}
```

（4）上传图片

- 在src/constants/api.js中定义常量映射到后端请求地址

```
export const API_USERIMAGES_ADD ='/api/v1/media/material/upload_picture'
```

- 在src/api/publish.js中定义请求方法

```
//上传图片
export function uploadImg (data) {
    return Request({
        url:API_USERIMAGES_ADD,
        method:'post',
        data
    })
}
```

4.6.2 路由调整

在src/router.js中asyncRouterMap对象的children数组中增加以下改动，以满足全局自动记录路由的功能：

```
{
    path:'/material/list',
    component: () => import('./views/material/material.vue'),
}
```

4.6.3 菜单调整

在src/constants/menus.js中的MenuData添加一下内容， 此处我们添加了我们之后将实现的所有菜单功能，此后将不再重复编写。

```
//导出菜单数据
export const MenuData = [
    {
        title:'首页',path : '/' ,icon:'el-icon-s-home'
    },
    {
        title:'内容管理',path:'/article',icon:'el-icon-edit',
        children:[
            { title:'图文数据' , path : '/material/data'},
            { title:'发布文章' , path : '/article/publish'},
            { title:'内容列表' , path : '/article/list'},
```

```
            { title:'评论列表' , path : '/comment/list'},
            { title:'素材管理' , path : '/material/list'}
        ]
    },
    {
        title:'粉丝管理', path:'/fans',icon:'el-icon-user',
        children:[
            { title:'粉丝概况' , path : '/fans/index'},
            { title:'粉丝画像' , path : '/fans/info'},
            { title:'粉丝列表' , path : '/fans/list'}
        ]
    },
    { title:'账户信息',path:'/user/center',icon:'el-icon-setting'}
]
```

4.6.4 实现素材管理页面

（1）上传组件定义

在src/components/Upload/中创建文件upload.vue，并实现一下代码

```
<template>
     <div class="upload_pic" >
              <el-form  status-icon label-width="100px">
                <img :src="upload_img_url" class="upload_pic_show" />
                <el-form-item label="用户图片" prop="logo">
                  <el-upload ref="myUpload" action="" :auto-upload="false">
                    <el-button size="small" type="primary">点击选择图片</el-
button>
                  </el-upload>
                </el-form-item>
                <el-form-item>
              <el-button type="primary" @click="fnUpload" size="small">开始上传
</el-button>
            </el-form-item>
        </el-form>
     </div>
</template>
<script>
import { uploadImg } from  '@/api/publish'
export default {
  name:"upload",
  props:['imgChange'],
  data () {
      return  {
         upload_img_url:require('@/assets/pic_bg.png'),
      }
  },
  methods:{
      //上传图片
     async fnUpload () {
        let files = document.querySelector('.el-upload .el-upload__input').files
;
        if(files && files.length) {
          let fd = new FormData();
          fd.append('file', files[0],files[0].name);
          let result = await uploadImg(fd)
```

```
            this.$message({message:'上传成功',type:'success'}) &&
(this.upload_img_url = result.url)
            this.imgChange && this.imgChange(result.url) //调用上层的方法 通知数据变化
          }else{
             this.$message({message:"请选择一张图片",type:"warning"})
          }
        }
    }
}
</script>

<style>
 .upload_pic_show{
    display:block;
    width:240px;
    height:180px;
    margin:15px auto 10px;
  }
</style>
```

（2）素材管理界面定义

在src/views/material/中定义material.vue，代码如下

```
<template>
 <div class="filter">
     <header>图片管理</header>
     <div class="container">
          <el-radio-group size='small' @change="loadData" v-model="activeSelect"
 style="margin-bottom: 30px;">
             <el-radio-button  label="0">全部</el-radio-button>
             <el-radio-button  label="1">收藏</el-radio-button>
          </el-radio-group>
          <el-button @click="showPicDialog = true" class="upload_btn"
type="primary">上传图片</el-button>
          <div class="img_list">
              <div class="img_list_item" v-for="img in imgData" :key="img.id">
                  <img :src="img.url" />
                  <div v-if="activeSelect == '0'" class="operate">
                     <img @click="collectOrCancel(img)" :src="img.is_collection
? collectSelectedIcon : collectIcon" alt="" />
                     <img @click="delImg(img)" :src="delIcon" alt="">
                  </div>
              </div>
          </div>
           <div class="pagination">
                <el-pagination
                      background
                      layout="prev, pager, next, jumper"
                      :total="imgPage.total"
                      :page-count="imgPage.pageCount"
                      :page-size="imgPage.pageSize"
                      :current-page="imgPage.currentPage"
                       @current-change="pageChange"
                       >
                </el-pagination>
              </div>
```

```
      </div>
        <el-dialog
          :visible.sync="showPicDialog"
           width="50%"
          :show-close="false"
          :center="true"
          :before-close="closeModal"
          title="上传图片">
            <upload v-if="showPicDialog" :imgChange="imgChangeCall" />
             <span slot="footer" class="dialog-footer">
              <el-button type="primary" @click="closeModal">关闭</el-button>
            </span>
       </el-dialog>
 </div>
 </template>
<script>
import { getAllImgData , delImg , collectOrCancel} from '@/api/publish'
import Upload from  '@/components/Upload/upload.vue'
export default {
    name:'material',
    data () {
        return {
            collectIcon:require('@/assets/collect.png'),//收藏图标
            collectSelectedIcon:require('@/assets/collect_select.png'),//收集图标
            delIcon:require('@/assets/del.png'),//删除图标
            imgPage:{
               total:0,
               currentPage:1,
               pageCount:0,
               pageSize:15
            },
            imgChange:false,//是否上传过图片导致图片数据变化 此状态用来控制是否在关闭后要进行重新加载
            showPicDialog:false,
            activeSelect:'0',
            imgData:[],//存储图片的数据 同时作为收藏数据和全部数据的引用
        }
    },
    components : {
        Upload
    },
    mounted () {
        this.loadData();
    },
    methods:{
      loadData : function(){
        //初始化时加载数据
        this.getImgData({
          page:this.imgPage.currentPage,
          size:this.imgPage.pageSize,
          is_collected:this.activeSelect
        })
      },
      //页面发生变化
      pageChange (newPage) {
        this.imgPage.currentPage = newPage
        this.loadData();
      },
```

```
      //获取图片素材
      async  getImgData (params) {
        let result = await getAllImgData(params)
        this.imgData = result.data.list
        this.imgPage.total = result.data.total
        this.imgPage.pageCount = Math.ceil(this.imgPage.total /
this.imgPage.pageSize)
      },
      //取消或者收藏图片
      async collectOrCancel (img) {
          let isCollected = img.is_collection;
          if(isCollected==1){ isCollected = 0; }else{ isCollected=1; }
          //取相反状态
         await collectOrCancel(img.id , {isCollected:isCollected})
         img.is_collection = isCollected //取相反状态
         this.$forceUpdate() //强制更新
         this.$message({type:'success',message:'操作成功'})
      },
      //删除图片
      async delImg (img) {
        let result =  await  this.$confirm('确认删除该素材?');
         result ? await delImg(img.id) : null //删除数据
        //写多了if  else 写个三元表达式 换换口味
         this.$message({type:'success',message:'删除成功'}) &&
         this.loadData();
      },
     imgChangeCall () {
         //图片变化了 记录改变的状态 用于关闭弹层时 重新加载数据
         this.imgChange = true
     },
     //关闭弹层时触发
     //注意 这里 为什么不在click用表达式赋值的方式去关掉弹层呢
     //因为发现在click="dialog = false" 模式下 不能触发关闭的回调 应该是实现机制的顺序问题
     closeModal () {
        if(this.imgChange){
           this.loadData()
           this.imgChange = false
        }
        this.showPicDialog = false
     }
  }
}
</script>
<style lang="scss" scoped>
 .filter {
    background-color: #ffffff;
    text-align: left;
    border: 1px solid #e7e7e9;
    header {
      border-bottom: 1px dashed #cccccc;
      margin: 0 5px;
      padding: 0 10px;
      font-size: 14px;
      height: 55px;
      line-height: 55px;
      color: #323745;
    }
    .container {
```

```
        padding: 20px;
        .upload_btn {
            position:absolute;
            right:40px;
            top:80px;
        }
    }
    .img_list {
        display: flex;
        flex-direction: row;
        flex-wrap: wrap;
        align-content: center;
        .img_list_item {
            margin:30px 40px;
            width: 150px;
            height:150px;
            position:relative;
            img {
                width: 100%;
                height:100%;
                border-radius: 5px;
            }
            .operate {
                position: absolute;
                width: 100%;
                height:30px;
                background: #f4f5f6;
                bottom: 0;
                left:0;
                display: flex;
                flex-direction: row;
                justify-content: space-around;
                align-items: center;
                img {
                    width: 16px;
                    height:16px;
                    cursor: pointer;
                }
            }
        }
    }
     .pagination {
            width: 100%;
            text-align: center
     }
 }
</style>
```

5 文章管理功能

5.1 文章发布、保存草稿后台接口

5.1.1接口定义

（1）基本定义

保存文章信息为草稿或发布文章

参考标准	请参考通用接口规范
发布接口名称	/api/v1/media/news/submit
草稿接口名称	/api/v1/media/news/submit
请求DTO	com.heima.model.media.dtos.WmNewsDto
响应DTO	{ "host": null, "code": 0, "error_message": "操作成功", "data": "SUCCESS" }

查询所有的channel

参考标准	请参考通用接口规范
发布接口名称	/api/v1/channel/channels
请求DTO	无
响应DTO	{ "host": null, "code": 0, "error_message": "操作成功", "data": List }

（2）CODE定义

PARAM_INVALID	PARAM_INVALID(501,"无效参数"),
PARAM_REQUIRE	PARAM_REQUIRE(500,"缺少参数")

(3)思路分析

- 如果用户传递参数为空或文章内容为空返回PARAM_REQUIRE错误
- 如果用户本次为修改操作那么先删除数据库中的信息
- 保存或修改文章的数据
- 保存内容中的图片和当前文章的关系
- 保存封面图片和当前文章的关系
- 流程处理完成返回处理结果

5.1.2 Mapper实现

（1）WmMaterialMapper

在类com.heima.model.mappers.wemedia.WmMaterialMapper中定义findMaterialByUidAndimgUrls方法用过用户id以及图片url查询所有的Material：

```
List<WmMaterial> findMaterialByUidAndimgUrls(Long uid,
Collection<Object> values);
```

WmMaterialMapper.xml

在文件resources/mappers/wemedia/WmMaterialMapper.xml，实现对应的SQL

```
<select id="findMaterialByUidAndimgUrls"
        resultType="com.heima.model.media.pojos.WmMaterial">
    select id, url
    from wm_material
    where user_id = #{uid}
    and url in
    <foreach item="item" index="index" collection="values"
             open="(" separator="," close=")">
        #{item}
    </foreach>
</select>
```

（2）WmNewsMaterialMapper

在com.heima.model.mappers.wemedia.WmNewsMaterialMapper接口中添加delByNewsId方法用于根据id删除文章、saveRelationsByContent用于保存文章和图片的关联关系。

```
int delByNewsId(Integer nid);
void saveRelationsByContent(Map<String, Object> materials, Integer newsId,
                            Short type);
```

WmNewsMaterialMapper.xml

在文件resources/ mappers/wemedia/WmNewsMaterialMapper.xml 中实现如下SQL如下：

```
<insert id="saveRelationsByContent">
    insert into wm_news_material (material_id, news_id, type, ord)
    values
    <foreach item="mid" index="ord" collection="materials.entrySet()"
             separator="," >
        (#{mid}, #{newsId}, #{type}, #{ord})
    </foreach>
</insert>

<delete id="delByNewsId">
    delete from wm_news_material
    where news_id = #{nid}
</delete>
```

（3）WmNewsMapper

WmNews实体类

```
@Data
public class WmNews {
    private Integer id;
    @IdEncrypt
    protected Long userId;
    private String title;
    private Short type;
    @IdEncrypt
    private Integer channelId;
    private String labels;
    private Date createdTime;
    private Date submitedTime;
    private Short status;
```

```
    private Date publishTime;
    private String reason;
    @IdEncrypt
    private Integer articleId;
    private String content;
    private String images; //图片用逗号分隔
}
```

在com.heima.model.mappers.wemedia.WmNewsMapper中添加接口方法insertNewsForEdit用于实现保存文章的操作、updateByPrimaryKey用于实现更新操作。

```
public interface WmNewsMapper {

    /**
     * 根据主键修改
     * @param dto
     * @return
     */
    int updateByPrimaryKey(WmNews record);
    /**
     * 添加草稿新闻
     * @param dto
     * @return
     */
    int insertNewsForEdit(WmNews dto);
}
```

WmNewsMapper.xml

```
<mapper namespace="com.heima.model.mappers.wemedia.WmNewsMapper">

    <resultMap id="BaseResultMap" type="com.heima.model.media.pojos.WmNews" >
        <id column="id"  property="id"/>
        <result column="user_id" property="userId"/>
        <result column="title" property="title"/>
        <result column="type" property="type"/>
        <result column="channel_id" property="channelId"/>
        <result column="labels" property="labels"/>
        <result column="created_time" property="createdTime"/>
        <result column="submited_time" property="submitedTime"/>
        <result column="status" property="status"/>
        <result column="publish_time" property="publishTime"/>
        <result column="reason" property="reason"/>
        <result column="article_id" property="articleId"/>
        <result column="content" property="content" jdbcType="LONGVARCHAR"
javaType="java.lang.String" />
        <result column="enable" property="enable" />
    </resultMap>
    <resultMap id="ResultMapWithBLOBs" type="com.heima.model.media.pojos.WmNews"
>
        <id column="id"  property="id"/>
        <result column="user_id" property="userId"/>
        <result column="title" property="title"/>
        <result column="type" property="type"/>
```

```
        <result column="channel_id" property="channelId"/>
        <result column="labels" property="labels"/>
        <result column="created_time" property="createdTime"/>
        <result column="submited_time" property="submitedTime"/>
        <result column="status" property="status"/>
        <result column="publish_time" property="publishTime"/>
        <result column="reason" property="reason"/>
        <result column="article_id" property="articleId"/>
        <result column="content" property="content" jdbcType="LONGVARCHAR"
javaType="java.lang.String" />
        <result column="enable" property="enable" />
    </resultMap>

    <sql id="Base_Column_List" >

    id, user_id,content, title, type, channel_id, labels, created_time,
submited_time, status,enable,
    publish_time, reason, article_id, images
  </sql>
    <update id="updateByPrimaryKey"
parameterType="com.heima.model.media.pojos.WmNews" >

    update wm_news
    set user_id = #{userId},
      title = #{title},
      type = #{type},
      channel_id = #{channelId},
      labels = #{labels},
      created_time = #{createdTime},
      submited_time = #{submitedTime},
      status = #{status,jdbcType=TINYINT},
      publish_time = #{publishTime},
      reason = #{reason},
      article_id = #{articleId},
      content = #{content},
      images = #{images}
    where id = #{id}
  </update>
    <insert id="insertNewsForEdit" useGeneratedKeys="true" keyProperty="id">
      insert  into wm_news(
        user_id, title, type, channel_id, labels, created_time, submited_time,
status,
        publish_time,content
      )
      values (
        #{userId},#{title},#{type},#{channelId},#{labels},#{createdTime},
        #{submitedTime},#{status},#{publishTime}, #{content}
      )
  </insert>

</mapper>
```

(4)获取所有的channel

AdChannel实体类

```
@Data
public class AdChannel {
    private Integer id;
    private String name;
    private String description;
    private Boolean isDefault;
    private Boolean status;
    private Byte ord;
    private Date createdTime;
}
```

定义AdChannelMapper接口：com.heima.model.mappers.admin.AdChannelMapper

```
public interface AdChannelMapper {
    /**
     * 查询所有
     */
    public List<AdChannel> selectAll();
}
```

AdChannelMapper.xml

```
<mapper namespace="com.heima.model.mappers.admin.AdChannelMapper">
    <resultMap id="BaseResultMap" type="com.heima.model.admin.pojos.AdChannel">
        <id column="id" property="id"/>
        <result column="name" property="name"/>
        <result column="description" property="description"/>
        <result column="is_default" property="isDefault"/>
        <result column="status" property="status"/>
        <result column="ord" property="ord"/>
        <result column="created_time" property="createdTime"/>
    </resultMap>
    <sql id="Base_Column_List">
    id, name, description, is_default, status, ord, created_time
  </sql>
    <sql id="Base_Column_where">
        <where>
            <if test="name!=null and name!=''">
                and name = #{name}
            </if>
            <if test="description!=null and description!=''">
                and description = #{description}
            </if>
            <if test="isDefault!=null and isDefault!=''">
                and is_default = #{isDefault}
            </if>
            <if test="status!=null and status!=''">
                and status = #{status}
            </if>
            <if test="ord!=null and ord!=''">
                and ord = #{ord}
            </if>
        </where>
    </sql>
    <!--查询所有频道-->
    <select id="selectAll" resultType="com.heima.model.admin.pojos.AdChannel">
```

```
        select
        <include refid="Base_Column_List"/>
        from ad_channel
    </select>
</mapper>
```

5.1.3 时序说明

- 如果用户传递参数为空或文章内容为空返回PARAM_REQUIRE错误
- 如果用户本次为修改操作那么先删除数据库关联数据
- 将用户提交的文章内容解析转为Map结构的数据
- 保存或修改文章的数据
- 保存内容中的图片和当前文章的关系
- 保存封面图片和当前文章的关系
- 流程处理完成返回处理结果

5.1.4 代码实现

（1）NewsService

创建类：com.heima.media.service.NewsService，并添加saveNews接口方法

```
public interface NewsService {
    /**
    * 自媒体发布文章
    * @param wmNews
    * @return
    */
    ResponseResult saveNews(WmNewsDto wmNews, Short type);
}
```

NewsServiceImpl

创建类：com.heima.media.service.impl.NewsServiceImpl并实现接口中的方法，在保存新闻的实现方法中分为以下步骤：

- 1.如果是修改先删除所有素材关联关系
- 2.解析文章类容，进行图文素材关联信息提取
- 3.保存发布文章信息
- 4.如果存在引用并且是提交审核则需要做关联，否则只是进行保存草稿则不进行内容素材关联操作
- 5.封面图片关联数据存储

```
@Service
@Slf4j
@SuppressWarnings("all")
public class NewsServiceImpl implements NewsService {

    @Autowired
    private ObjectMapper objectMapper;

    @Autowired
    private WmMaterialMapper wmMaterialMapper;

    @Autowired
```

```
    private WmNewsMapper wmNewsMapper;

    @Autowired
    private WmNewsMaterialMapper wmNewsMaterialMapper;

    @Autowired
    private ApArticleConfigMapper apArticleConfigMapper;

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

    @Override
    public ResponseResult saveNews(WmNewsDto dto, Short type) {
        if (dto == null || !StringUtils.isNotEmpty(dto.getContent())) {
            return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_INVALID);
        }
        WmUser user = WmThreadLocalUtils.getUser();
        //如果是修改先删除所有素材关联关系
        if (dto.getId() != null){
            wmNewsMaterialMapper.delByNewsId(dto.getId());
        }

        //解析文章类容，进行图文素材关联
        String content = dto.getContent();
        //Map<图片排序号， dfs文件id>
        Map<String, Object> materials;
        try {
            List<Map> list = objectMapper.readValue(content, List.class);
            //抽取信息
            Map<String, Object> extractInfo = extractUrlInfo(list);
            materials = (Map<String, Object>) extractInfo.get("materials");
            //文章图片总数量
            int countImageNum = (int) extractInfo.get("countImageNum");
            //保存发布文章信息
            WmNews wmNews = new WmNews();
            BeanUtils.copyProperties(dto, wmNews);
            if (dto.getType().equals(WmMediaConstans.WM_NEWS_TYPE_AUTO)){
                saveWmNews(wmNews, countImageNum, type);
            }else{
                saveWmNews(wmNews, dto.getType(), type);
            }
            //保存内容中的图片和当前文章的关系
            if (materials.keySet().size() != 0) {
                ResponseResult responseResult =
saveRelativeInfoForContent(materials, wmNews.getId());
                if (responseResult != null) {
                    return responseResult;
                }
            }
            //封面图片关联
            ResponseResult responseResult = coverImagesRelation(dto, materials,
wmNews, countImageNum);
            if (responseResult != null) {
                return responseResult;
            }
        } catch (IOException e) {
            e.printStackTrace();
            log.error("parse content error, param content :{}", content);
```

```
            return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_INVALID);
        }
        return ResponseResult.okResult(AppHttpCodeEnum.SUCCESS);
    }

    /**
     * 封面图片关联
     * @param dto
     * @param materials
     * @param wmNews
     * @param countImageNum
     * @return
     */
    private ResponseResult coverImagesRelation(WmNewsDto dto, Map<String,
Object> materials, WmNews wmNews, int countImageNum) {
        List<String> images = dto.getImages();
        if (!WmMediaConstans.WM_NEWS_TYPE_AUTO.equals(dto.getType()) &&
dto.getType() != images.size()) {
            return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_INVALID, "图
文模式不匹配");
        }
        //如果是自动匹配封面
        if (WmMediaConstans.WM_NEWS_TYPE_AUTO.equals(dto.getType())) {
            images = new ArrayList<>();
            if (countImageNum == WmMediaConstans.WM_NEWS_SINGLE_IMAGE) {
                for (Object value : materials.values()) {
                    images.add(String.valueOf(value));
                    break;
                }
            }
            if (countImageNum >= WmMediaConstans.WM_NEWS_MANY_IMAGE) {
                for (int i = 0; i < WmMediaConstans.WM_NEWS_MANY_IMAGE; i++) {
                    images.add((String) materials.get(String.valueOf(i)));
                }
            }
            if (images.size() != 0) {
                ResponseResult responseResult = saveRelativeInfoForCover(images,
wmNews.getId());
                if (responseResult != null) {
                    return responseResult;
                }
            }
        } else if(images != null && images.size() != 0) {
            ResponseResult responseResult = saveRelativeInfoForCover(images,
wmNews.getId());
            if (responseResult != null) {
                return responseResult;
            }
        }
        //更新images字段
        if (images != null) {
            wmNews.setImages(
                    StringUtils.join(
                            images.stream().map(s -> s.replace(fileServerUrl,
"")).collect(Collectors.toList()),
                            WmMediaConstans.WM_NEWS_IMAGES_SWPARATOR
                    )
            );
```

```
            wmNewsMapper.updateByPrimaryKey(wmNews);
        }
        return null;
    }

    /**
     * 提取信息
     * @param list
     * @return
     */
    private Map<String, Object> extractUrlInfo(List<Map> list) {
        Map<String, Object> res = new HashMap<>();
        Map<String, Object> materials = new HashMap<>();
        int order = 0;
        int countImageNum = 0;
        //收集文章中引用的资源服务器的图片url以及排序
        for (Map map : list) {
            order++;
            if (WmMediaConstans.WM_NEWS_TYPE_IMAGE.equals(map.get("type"))) {
                countImageNum++;
                String imgUrl = String.valueOf(map.get("value"));
                if(imgUrl.startsWith(fileServerUrl)) {
                    materials.put(String.valueOf(order),
imgUrl.replace(fileServerUrl, ""));
                }
            }
        }
        res.put("materials", materials);
        res.put("countImageNum", countImageNum);
        return res;
    }

    /**
     * 保存关联信息到数据库
     * @param materials
     * @param newsId
     */
    private ResponseResult saveRelativeInfo(Map<String, Object> materials,
Integer newsId, Short type) {
        WmUser user = WmThreadLocalUtils.getUser();
        //手机数据库中的素材信息
        List<WmMaterial> dbMaterialInfos =
wmMaterialMapper.findMaterialByUidAndimgUrls(user.getId(), materials.values());
        if (dbMaterialInfos != null && dbMaterialInfos.size() != 0) {
            Map<String, Object> urlIdMap =
dbMaterialInfos.stream().collect(Collectors.toMap(WmMaterial::getUrl,
WmMaterial::getId));
            for (String key : materials.keySet()) {
                String fileId =
String.valueOf(urlIdMap.get(materials.get(key)));
                if ("null".equals(fileId)) {
                    return
ResponseResult.errorResult(AppHttpCodeEnum.PARAM_INVALID, "应用图片失效");
                }
                materials.put(key, String.valueOf(fileId));
            }
            //存储关系数据到数据库
```

```
            wmNewsMaterialMapper.saveRelationsByContent(materials, newsId,
type);
        }
        return null;
    }


    /**
     * 保存图片关系为封面
     * @param images
     * @param newsId
     */
    private ResponseResult saveRelativeInfoForCover(List<String> images, Integer
newsId) {
        Map<String, Object> materials = new HashMap<>();
        for (int i = 0; i < images.size(); i++) {
            String s = images.get(i);
            s = s.replace(fileServerUrl, "");
            materials.put(String.valueOf(i), s);
        }
        return saveRelativeInfo(materials, newsId,
WmMediaConstans.WM_IMAGE_REFERENCE);
    }


    /**
     * 保存图片关系为内容
     * @param materials
     * @param newsId
     */
    private ResponseResult saveRelativeInfoForContent(Map<String, Object>
materials, Integer newsId) {
        return saveRelativeInfo(materials, newsId,
WmMediaConstans.WM_CONTENT_REFERENCE);
    }

    /**
     * 保存/修改发布文章信息
     * @param wmNews
     * @param countImageNum
     * @param type
     */
    private void saveWmNews(WmNews wmNews, int countImageNum, Short type) {
        WmUser user = WmThreadLocalUtils.getUser();
        //保存提交文章数据
        if (countImageNum == WmMediaConstans.WM_NEWS_SINGLE_IMAGE) {
            wmNews.setType(WmMediaConstans.WM_NEWS_SINGLE_IMAGE);
        } else if (countImageNum >= WmMediaConstans.WM_NEWS_MANY_IMAGE) {
            wmNews.setType(WmMediaConstans.WM_NEWS_MANY_IMAGE);
        } else {
            wmNews.setType(WmMediaConstans.WM_NEWS_NONE_IMAGE);
        }
        wmNews.setStatus(type);
        wmNews.setUserId(user.getId());
        wmNews.setCreatedTime(new Date());
        wmNews.setSubmitedTime(new Date());
        wmNews.setEnable((short)1);
        if (wmNews.getId() == null) {
```

```
            wmNewsMapper.insertNewsForEdit(wmNews);
        }else {
            wmNewsMapper.updateByPrimaryKey(wmNews);
        }

    }

}
```

(2)查询所有的channel，定义接口：com.heima.media.service.AdChannelService

```
public interface AdChannelService {
    List<AdChannel> selectAll();
}
```

AdChannelServiceImpl实现类

```
@Service
public class AdChannelServiceImpl implements AdChannelService {

    @Autowired
    private AdChannelMapper channelMapper;

    @Override
    public List<AdChannel> selectAll() {
        return channelMapper.selectAll();
    }
}
```

（3）WmNewsDto

创建类：com.heima.model.media.dtos.WmNewsDto

此类在model模块中创建，定义请求入参，实现如下：

```
@Data
public class WmNewsDto {
    private Integer id;
    private String title;
    @IdEncrypt
    private Integer channelId;
    private String labels;
    private Date publishTime;
    private String content;
    private Short type;
    private Date submitedTime;
    private Short status;
    private String reason;
    private List<String> images;
}
```

（4）NewsControllerApi

在类com.heima.media.apis.NewsControllerApi中增加summitNews、saveDraftNews方法

```
public interface NewsControllerApi {

   /**
    * 提交文章*
    * @param wmNews*
    * @return*
    */
   ResponseResult summitNews(WmNewsDto wmNews);

    /**
    * 保存草稿
    * @param wmNews
    * @return
    */
    ResponseResult saveDraftNews(WmNewsDto wmNews);

}
```

（5）NewsController

在com.heima.media.controller.v1.NewsController类中实现NewsControllerApi接口方法，调用对应的service接口即可。

```
@RestController
@RequestMapping("/api/v1/media/news")
public class NewsController implements NewsControllerApi {

    @Autowired
    private NewsService newsService;

    @PostMapping("/submit")
    @Override
    public ResponseResult summitNews(@RequestBody WmNewsDto wmNews) {
        return
newsService.saveNews(wmNews,WmMediaConstans.WM_NEWS_SUMMIT_STATUS);
    }


    @PostMapping("/save_draft")
    @Override
    public ResponseResult saveDraftNews(@RequestBody WmNewsDto wmNews) {
        return newsService.saveNews(wmNews,
WmMediaConstans.WM_NEWS_DRAFT_STATUS);
    }

}
```

(6)定义api接口：com.heima.media.apis.AdChannelControllerApi

```
public interface AdChannelControllerApi {
    public ResponseResult selectAll();
}
```

(7)AdChannelController

```
@RestController
@RequestMapping("/api/v1/channel")
public class ChannelController implements AdChannelControllerApi {
    @Autowired
    private AdChannelService channelService ;
    @Override
    @RequestMapping("/channels")
    public ResponseResult selectAll(){
        return ResponseResult.okResult(channelService.selectAll());
    }
}
```

???经过测试，别管是草稿还是发布审核状态都是1，草稿的状态应该为0

5.2 文章发布，保存草稿前台

5.2.1 接口定义

（1）获取所有频道

- 在src/constants/api.js中定义常量映射到后端请求地址

```
export const API_CHANNELS = '/api/v1/channel/channels' //获取文章频道
```

- 在src/api/publish.js中定义请求方法（此处省略了引入刚才定义的常量，此后所有导入省略请自行导入需要的常量及方法）

```
export function getChannels () {
    return Request({
        url:API_CHANNELS,
        method:'get',
    })
}
```

（2）发布文章

- 在src/constants/api.js中定义常量映射到后端请求地址

```
export const API_ARTICLES = '/api/v1/media/news/submit' //post文章(新建)
```

- 在src/api/publish.js中定义请求方法

```
//发表文章
export function publishArticles (params,data) {
    console.log(params,data)
    return Request({
        url:API_ARTICLES,
        method:'post',
        params,
        data
    })
}
```

（3）修改文章

- 在src/constants/api.js中定义常量映射到后端请求地址

```
export const API_ARTICLES = '/api/v1/media/news/submit' //post文章(新建)
```

- 在src/api/publish.js中定义请求方法

```
//编辑文章
export function modifyArticles (articleId,params,data) {
    return Request({
        url:API_ARTICLES,
        method:'post',
        params,
        data
    })
}
```

（4）根据ID获取文章

- 在src/constants/api.js中定义常量映射到后端请求地址

```
export const API_ARTICLES_INFO = '/api/v1/media/news/news' //获取文章
```

- 在src/api/content.js中定义请求方法

```
//获取文章
export function getArticleById (articlesId) {
    return Request({
        url:API_ARTICLES_INFO,
        method:'post',
        params:{},
        data:{id:articlesId}
    })
}
```

5.2.2 路由调整

在src/router.js中asyncRouterMap对象的children数组中增加以下改动，以满足全局自动记录路由的功能：

```
{
    path:'/article/publish',
    component: () => import('@/views/publish/index.vue'),
}
```

5.2.3 实现文章发布页面

文章内容要在不同平台上通用解析，直接使用html富文本编辑器做解析，实现成本较为之大，在黑马项目中通过JSON数组来存储文章内容数据，一个元素就是一段内容，支持文字、图片等混排以及样式的调整。

（1）黑马编辑器组件定义

在src/components/editor/中创建文件heima.vue， 并实现一下代码

```
<template>
  <div class="item-wapper">
    <div class="item" v-for="(item,key) in content">
      <div class="item-t-bar">
        <li @click="deleteItem(key)" title="删除">&#xf00d;</li>
        <li @click="clearStyle(key)" title="恢复样式"  v-
if="item.type=='text'">&#xf122;</li>
        <li @click="enditorText(key)" title="编辑内容"  v-
if="item.type=='text'">&#xf044;</li>
        <li @click="editImg(key)" title="重新选择"  v-
if="item.type=='image'">&#xf044;</li>
        <li @click="bold(key)" title="加粗"  v-if="item.type=='text'">&#xf032;
</li>
        <li @click="up(key)" title="上移">&#xf062;</li>
        <li @click="down(key)" title="下移">&#xf063;</li>
        <li @click="upFontSize(key)" title="加大字号"  v-
if="item.type=='text'">&#xf15d;</li>
        <li @click="downFontSize(key)" title="减小字号"  v-
if="item.type=='text'">&#xf15e;</li>
        <li @click="addText(key)" title="添加文字" style="float: left">&#xf034;
</li>
        <li @click="addImg(key)" title="添加图片" style="float: left">&#xf03e;
</li>
      </div>
      <div class="item-t" v-if="item.type=='text'" :style="item.style">
{{item.value}}</div>
      <div class="item-i" v-if="item.type=='image'"><img style="max-width:
290px" :src="item.value"></div>
    </div>
    <el-dialog :title="controller.editorTitle"
:visible.sync="controller.dialogTextVisible">
      <el-form>
        <el-input
          type="textarea"
          :rows="5"
          placeholder="请输入内容"
          v-model="controller.editorText">
        </el-input>
      </el-form>
      <div slot="footer" class="dialog-footer">
        <el-button @click="saveText('cancel')">取 消</el-button>
        <el-button type="primary" @click="saveText('ok')">确 定</el-button>
      </div>
    </el-dialog>
  </div>
</template>

<script>
  export default {
    name: "heima",
    props:{
      datas : {
        type:Array,
        default:function () {
          return [
            {
              type: 'text',
```

```
            value: '请输入文章内容...'
          }
        ]
      }
    }
  },
  data(){
    return {
      text:{
        addText:'添加文字',
        editText:'编辑文字'
      },
      controller:{
        editorKey : 0,
        editorTitle:'',
        editorText : "",
        dialogTextVisible:false
      },
      content:[]
    }
  },
  created(){
    this.setContent(this.datas)
  },
  methods:{
    setContent : function(data){
      if(data.length>0){
        this.content = data;
      }else{
        this.content = this.datas
      }
    },
    deleteItem:function(key){
      if(this.content.length>1) {
        this.$confirm('删除后不可找回，确认删除吗?', '提示', {
          confirmButtonText: '确定',
          cancelButtonText: '取消',
          type: 'warning'
        }).then(() => {
          this.content.splice(key, 1)
          this.$message({type: 'success', message: '删除成功!'});
        })
      }else{
        this.$message({ type: "warning", message: "不能全部删除内容，请编辑！" });
      }
    },
    clearStyle:function(key){
      this.getStyle(key,'w',"0")
      this.$set(this.content[key],'style',{})
    },
    // 编辑
    editImg : function(key){
      this.$emit("addImg",{key:key,type:'edit'});
    },
    // 增加或者删除
    addImg : function(key){
      this.$emit("addImg",{key:key,type:'add'});
    },
```

```
      // 添加文字
      addText:function(key){
        this.controller.editorTitle=this.text.addText
        this.controller.editorKey=key
        this.controller.editorText=''
        this.controller.dialogTextVisible=true
      },
      // 编辑文本
      enditorText:function(key){
        this.controller.editorTitle=this.text.editText
        this.controller.editorKey=key
        this.controller.editorText=this.content[key].value
        this.controller.dialogTextVisible=true
      },
      // 保存图片
      saveImage: function(data,image){
        if(data.type=='add'){
          let value = {
            type:'image',
            value:image
          }
          this.content.splice(data.key,0,value)
        }else{
          this.$set(this.content[data.key], 'type', "image")
          this.$set(this.content[data.key], 'value', image)
        }
      },
      // 保存编辑和新增的文字
      saveText: function(button){
        if(button=='ok'){
          if(this.controller.editorText!='') {
            if (this.controller.editorTitle == this.text.editText) {
              this.$set(this.content[this.controller.editorKey], 'value',
this.controller.editorText)
              this.controller.dialogTextVisible = false
            }else{
              this.content.splice(this.controller.editorKey,0,
{type:'text',value:this.controller.editorText});
              this.controller.dialogTextVisible = false
            }
          }else{
            alert('文字内容不能为空！')
          }
        }else{
          this.controller.dialogTextVisible=false
        }
      },
      bold:function (key) {
        let temp = this.getStyle(key,'fontWeight',"normal")
        if(temp!='bold'){temp='bold';}else{temp='normal';}
        this.$set(this.content[key]['style'],'fontWeight',temp)
      },
      up : function(key){
        let i = key-1
        if(i>=0){
          this.content[i] = this.content.splice(key, 1, this.content[i])[0];
        }
      },
```

```
      down : function(key){
        let i = key+1
        if(i<this.content.length){
          this.content[i] = this.content.splice(key, 1, this.content[i])[0];
        }
      },
      upFontSize:function (key) {
        let temp = this.getStyle(key,'fontSize','12')
        this.$set(this.content[key]['style'],'fontSize',(parseInt(temp)+1)+"px")
      },
      downFontSize:function (key) {
        let temp = this.getStyle(key,'fontSize','12')
        this.$set(this.content[key]['style'],'fontSize',(parseInt(temp)-1)+"px")
      },
      // 获取一个样式
      getStyle:function(key,name,defValue){
        let style=this.content[key]['style']
        if(style==undefined){
          style = {}
          this.$set(this.content[key],'style',style)
        }
        let temp = style[name]
        if(temp==undefined){
          temp=defValue
          this.$set(this.content[key]['style'],name,defValue)
        }else{
          temp=temp.toLowerCase()
        }
        return temp.replace(';','').replace('px','')
      },
      // 过滤空样式
      getItemStyle:function (style) {
        if(style!=undefined){
          return style
        }
        return {}
      },
      getContent:function () {
        return JSON.stringify(this.content)
      }
    }
  }
</script>

<style scoped>
  .item-wapper{
    border: 1px solid #dbdbdb;
    width: 310px;
    max-height: 550px;
    overflow-x: hidden;
    overflow-y: auto;
    padding: 15px 10px;
    border-radius: 10px;
  }
  .item-wapper::-webkit-scrollbar {/*滚动条整体样式*/
    width: 10px;     /*高宽分别对应横竖滚动条的尺寸*/
    height: 1px;
  }
```

```
  .item-wapper::-webkit-scrollbar-thumb {/*滚动条里面小方块*/
    -webkit-box-shadow: inset 0 0 5px rgba(0,0,0,0.2);
    border-top-right-radius: 10px;
    border-bottom-right-radius: 10px;
    background: #dbdbdb;
  }
  .item-wapper::-webkit-scrollbar-track {/*滚动条里面轨道*/
    background: transparent;
  }
  .item{
    position: relative;
    overflow: hidden;
    text-align: left;
    border: 1px solid #ffffff;
  }
  .item-t{
    min-height: 30px;
    font-size: 12px;
    line-height: 150%;
    margin: 5px 0px;
  }
  .item-i{
    margin: 5px 0px;
  }
  .item-i img{
    padding: 0px;
    margin: 0px;
    display: block;
    border: none;
  }
  .item:hover{
    border: 1px solid #dbdbdb;
    border-radius: 10px;
  }
  .item:hover .item-t-bar{
    display: inherit;
  }
  .item-t-bar{
    display: none;
    position: absolute;
    background-color: red;
    opacity: 0.9;
    width: 100%;
    color: #FFFFFF;
    z-index: 999;
    overflow: hidden;
  }
  .item-t-bar li{
    list-style: none;
    float: right;
    line-height: 30px;
    background-color: red;
    font-size: 10px;
    font-family: "FontAwesome";
    padding: 0px 5px;
    cursor: pointer;
  }
</style>
```

（2）发布页面实现

发布页面就是基本的VUE表单页面，在src/views/publish/index.vue文件中实现如下：

```
<template>
  <div class="tinymce-container">
    <header>发表文章</header>
    <el-form ref="form" label-width="120px">
      <el-form-item label="标题" prop="title">
        <el-input
          v-model="FormData.title"
          placeholder="文章名称"
          style="width: 400px;"
          class="filter-item"
        />
      </el-form-item>
      <el-form-item label="标签" prop="labels">
        <el-input
          v-model="FormData.labels"
          placeholder="内容标签"
          style="width: 400px;"
          class="filter-item"
        />
      </el-form-item>
      <el-form-item label="频道：" prop="channel_id">
        <el-select v-model="FormData.channel_id" size="small" style="width:
400px;">
          <el-option
            v-for="item in channel_list"
            :key="item.id"
            :label="item.name"
            :value="item.id"
          ></el-option>
        </el-select>
      </el-form-item>
      <el-form-item label="定时：" prop="publish_time">
        <el-date-picker
          v-model="FormData.publish_time"
          type="datetime"
          style="width: 400px;"
          placeholder="选择日期时间"
          default-time="12:00:00">
        </el-date-picker>
      </el-form-item>
      <el-form-item label="内容">
        <Heima ref="heima" @addImg="selectHeiMaImg"/>
      </el-form-item>
      <el-form-item label="封面">
        <el-radio-group v-model="FormData.type">
          <el-radio label="1">单图</el-radio>
          <el-radio label="3">三图</el-radio>
          <el-radio label="0">无图</el-radio>
          <el-radio label="-1">自动</el-radio>
        </el-radio-group>
      </el-form-item>
      <el-form-item v-if="FormData.type == '1' || FormData.type == '3'">
```

```
        <div v-if="FormData.type == '1'" class="single_pic"
@click="selectSinglePic">
          <div class="title">点击图标选择图片</div>
          <img :src="parseImage(singlePic)">
        </div>
        <div v-if="FormData.type == '3'" class="three_pic">
          <div
            class="three_pic_item"
            v-for="(item,index) in threePicList"
            :key="index"
            @click="selectThreePic(index)"
          >
            <div class="title">点击图标选择图片</div>
            <img :src="parseImage(item)">
          </div>
        </div>
      </el-form-item>
      <el-form-item class="btn">
        <el-button @click="publish(false)" class="filter-item" type="primary">提
交审核</el-button>
        <el-button @click="publish(true)" class="filter-item">存入草稿</el-
button>
      </el-form-item>
    </el-form>
    <el-dialog
      :visible.sync="showPicDialog"
      width="50%"
      :close-on-click-modal="false"
      :show-close="false"
      :center="true"
    >
      <el-tabs type="card" v-model="activeName">
        <el-tab-pane label="素材库" name="first">
          <el-radio-group @change="getImgData" v-model="activeName2"
style="margin-bottom: 30px;">
            <el-radio-button label="all">全部</el-radio-button>
            <el-radio-button label="collect">收藏</el-radio-button>
          </el-radio-group>
          <div class="img_list_con">
            <div
              class="img_list"
              v-for="item in imgData"
              :key="item.id"
              @click="selectPic(item.id,item.url)"
            >
              <img :src="item.url">
              <img v-if="item.id == selectedImg.id" :src="selected_img_url"
class="selected">
            </div>
          </div>
          <div class="pagination">
            <el-pagination
              background
              layout="total, prev, pager, next, jumper"
              :page-size="imgPage.pageSize"
              :total="imgPage.total"
              :page-count="imgPage.pageCount"
              :current-page.sync="imgPage.currentPage"
```

```
              @current-change="getImgData"
            ></el-pagination>
          </div>
        </el-tab-pane>
        <el-tab-pane label="上传图片" name="second">
          <upload :imgChange="uploadSuccess"/>
        </el-tab-pane>
      </el-tabs>
      <span slot="footer" class="dialog-footer">
        <el-button @click="cancleImg">取 消</el-button>
        <el-button type="primary" @click="btnOKImg">确 定</el-button>
      </span>
    </el-dialog>
  </div>
</template>

<script>
  import Heima from "@/components/editor/heima.vue";
  import Upload from "@/components/Upload/upload.vue";
  import { getArticleById } from "@/api/content";
  import {
    getAllImgData,
    getChannels,
    publishArticles,
    modifyArticles
  } from "@/api/publish";
  export default {
    name: "HeiMa",
    components: { Upload,Heima },
    data() {
      return {
        FormData: {
          id:"",
          title: "", //标题
          type: "3",
          labels:"",
          publish_time:"",
          channel_id: null//频道ID
        },
        host:'',//图片host
        singlePic: null, //单图模式
        threePicList: [null, null, null], //三图模式
        pubForm: {},
        channel_list: [],
        showPicDialog: false, //显示图片上传提示框
        activeName: "first",
        activeName2: "all",
        selected_img_url: require("@/assets/selected.png"),
        upload_img_url: require("@/assets/pic_bg.png"),
        imgPage: {
          /***用来存储页面的页码及行数信息*****/
          total: 0, //总页数
          currentPage: 1, //第几页
          pageSize: 5, //每页多少条
          pageCount: 1 //共多少页
        },
        imgData: [], //存储图片的数据
        selectedImg: {}, //已经选择的图片
```

```
        currentType: {
          key:0,//编辑序列
          type: "" //存储弹层的操作类型  single three insert  之所以用对象是因为要存放三张图的索引
        }
      };
    },
    beforeMount() {
      let { articleId } = this.$route.query;
      if (articleId) {
        //如果id存在 则拉取新数据
        this.getArticle(articleId);
      }
      this.getChannels(); //拉取文章频道
    },
    methods: {
      parseImage : function(item){
        if(item){
          if(item.indexOf('http')>-1){
            return item;
          }else{
            return this.host+item;
          }
        }else{
          return this.upload_img_url
        }
      },
      //获取文章频道
      async getChannels() {
        let result = await getChannels();
        this.channel_list = result.data;
      },
      //获取文章
      async getArticle(id) {
        let result = await getArticleById(id);
        this.FormData = {
          id:result.data.id,
          title: result.data.title,
          channel_id: result.data.channel_id,
          labels:result.data.labels,
          type: ""+result.data.type,
          publish_time:result.data.publish_time
        }
        let conts = [];
        if(result.data.content){
          try{
            conts = eval("("+result.data.content+")")
          }catch (e) {
            console.error(e)
          }
        }
        this.$refs.heima.setContent(conts)
        this.host = result.host
        this.transImages(this.FormData.type, result.data.images); //还原数据
      },
      //选择一张图片
      selectPic(id, url) {
        this.selectedImg = {id,url };
```

```
      },
      //上传成功后
      uploadSuccess(url) {
        this.selectedImg = { url }; //将上传的图片认为是新组件
      },
      selectHeiMaImg(key){
        this.currentType.key = key;
        this.currentType.type="insert"
        this.uploadPic();
      },
      //点击图片上传图标
      uploadPic() {
        this.imgPage.currentPage=1
        this.showPicDialog = true; //显示弹层
        this.getImgData(); //拉取图片数据
      },
      //插入图片 或者替换封面图片
      btnOKImg() {
        if (this.selectedImg.url) {
          if(this.selectedImg.url.indexOf('http')>0){
            this.selectedImg.url=this.host+this.selectedImg.url;
          }
          if (this.currentType.type == "single") {
            this.singlePic = this.selectedImg.url;
          } else if (this.currentType.type == "three") {
            //三张图时  需要找到数组中存储的对象
            this.threePicList[this.currentType.index] = this.selectedImg.url; //
找到那条记录更新
            this.$forceUpdate(); //由于直接改变的对象 所以这里强制更新下
          } else if (this.currentType.type == "insert") {

 this.$refs['heima'].saveImage(this.currentType.key,this.selectedImg.url)
          }
        }
        this.currentType = {}; //清空类型缓存
        this.selectedImg = {}; //首先清空选择的缓存
        this.showPicDialog = false; //关闭弹层
      },
      //取消插入
      cancleImg() {
        this.showPicDialog = false; //关闭弹层
      },
      //点击三图中的图片
      selectThreePic(index) {
        this.currentType.type = "three";
        this.currentType.index = index; //这里需要记录图片的索引 因为要按照顺序 不能乱
        this.uploadPic(); //打开弹层
      },
      //选择单张图片
      selectSinglePic() {
        this.currentType.type = "single";
        this.uploadPic(); //打开弹层
      },
      //拉取所有的图片数据
      async getImgData(page) {
        let temp = page==undefined?this.imgPage.currentPage:page
        try{
          temp = parseInt(temp)
```

```
      }catch (e) {
        temp=1
      }
      let isCollect = this.activeName2 == "collect"; //是否是收藏的列表
      let result = await getAllImgData({
        size: this.imgPage.pageSize,
        page: temp,
        is_collected: isCollect?1:0 //是否是收藏
      });
      this.imgData = result.data.list;
      this.imgPage.total = result.data.total;
      this.imgPage.pageCount = Math.ceil(
        this.imgPage.total / this.imgPage.pageSize
      );
    },
    //转换图片
    transImages(type, images) {
      images=images.split(",")
      if (type == "1") {
        this.singlePic = images[0];
      } else if (type == "3") {
        this.threePicList = [...images];
      }
    },
    //获取图片列表
    getImages() {
      if (
        this.FormData.type == "1" ||
        this.FormData.type == "3"
      ) {
        if (this.FormData.type == "1") {
          return this.singlePic ? [this.singlePic] : [];
        } else {
          return this.threePicList.map(item => item);
        }
      }
      return [];
    },
    //发布文章
    async publish(draft) {
      let { articleId } = this.$route.query;
      let params = {draft}; //路径参数
      let images = this.getImages();
      let data = {
        ...this.FormData,
        images: images,
        status:draft?0:1,
        content:this.$refs.heima.getContent()
      }; //请求参数
      if (!draft) {
        //非草稿需要校验
        if (!data.title || data.title.length < 5||data.title.length>32) {
          this.$message({
            type: "warning",
            message: "文章标题不能小于5个字符或大于32个字符"
          });
          return;
        }
```

```
          if (!data.labels||data.title.length>20) {
            this.$message({ type: "warning", message: "内容标签不能为或超过20字符"
});
            return;
          }
          if (!data.content) {
            this.$message({ type: "warning", message: "文章内容不能为空" });
            return;
          }
          if (!data.channel_id) {
            this.$message({ type: "warning", message: "文章频道不能为空" });
            return;
          }
          if (
            (data.type == "1" && data.images.length != 1) ||
            (data.type == "3" && data.images.length != 3)
          ) {
            this.$message({ type: "warning", message: "文章封面未设置" });
            return;
          }
        }
        //编辑或者发布文章
        !articleId
          ? await publishArticles(params, data)
          : await modifyArticles(articleId, params, data);
        this.$message({
          type: "success",
          message: articleId ? "编辑文章成功" : "新增文章成功"
        });
        this.$router.replace({ path: "/article/list" });
      }
    }
  };
</script>

<style rel="stylesheet/scss" lang="scss" scoped>
  .tinymce-container {
    background-color: #ffffff;
    text-align: left;
    border: 1px solid #e7e7e9;
  header {
    color: #323745;
    font-size: 14px;
    height: 55px;
    line-height: 55px;
    padding: 0 15px;
    background-color: #fbfbfb;
    border-bottom: 1px solid #e8e8e8;
  }
  .el-form {
    padding: 20px 30px 0 0;
  .el-form-item {
    margin: 20px 0;
  }
  .btn {
    border-top: 1px solid #e8e8e8;
    margin: 0 15px;
    padding: 30px 0;
```

```
  }
  }
  }
  .editor-content {
    margin-top: 20px;
  }
  .img_list {
    width: 128px;
    height: 100px;
    float: left;
    margin: 0px auto;
    border: 1px solid #eee;
    overflow: hidden;
    border-radius: 4px;
    margin: 0px 20px 20px 0;
    position: relative;
  }
  .img_list_con {
    overflow: hidden;
    margin-left: 20px;
    height: 250px;
  }
  .selected {
    width: 60px !important;
    height: 60px !important;
    position: absolute;
    bottom: 0;
    left: 0;
    margin-left: 50%;
    margin-bottom: 50%;
    transform: translate(-30px, 50px);
  }
  .img_list img {
    width: 128px;
    height: 100px;
    display: block;
    cursor: pointer;
  }
  .pagination {
    text-align: center;
  }
  .upload_pic_show {
    display: block;
    width: 240px;
    height: 180px;
    margin: 15px auto 10px;
  }
  .single_pic {
    border: 1px solid #cccccc;
    width: 280px;
    height: 280px;
    border-radius: 4px;
  .title {
    text-align: center;
  }
  img {
    width: 220px;
    height: 220px;
```

```
    margin: 0 auto;
    display: block;
  }
  }
  .three_pic {
    display: flex;
    flex-direction: row;
    border: 1px solid #cccccc;
    width: 840px;
  .three_pic_item {
    width: 280px !important;
  .title {
    text-align: center;
  }
  img {
    width: 220px;
    height: 220px;
    margin: 0 auto;
    display: block;
  }
  }
  }
</style>
```

5.3 文章列表后台接口

5.3.1 接口定义

（1）基本定义

由于框架封装只对JSON反序列化自增ID，需要请求文章ID需要封装为DTO.

参考标准	请参考通用接口规范
发布接口名称	/api/v1/media/news/list
请求DTO	com.heima.model.media.dtos.WmNewsPageReqDto
响应DTO	{"host":,"code": 0,"error_message": "操作成功","data": []}

（2）CODE定义

PARAM_INVALID	PARAM_INVALID(501,"无效参数"),

5.3.2 Mapper实现

（1）WmNewsMapper

在类com.heima.model.mappers.wemedia.WmNewsMapper中定义selectBySelective、countSelectBySelective实现分页查询

```
public interface WmNewsMapper {

    /**
```

```
    * 查询根据dto条件
    * @param dto
    * @param uid
    * @return
    */
    List<WmNews> selectBySelective(WmNewsPageReqDto dto, Long uid);

    /**
    * 查询总数统计
    * @param dto
    * @param uid
    * @return
    */
    int countSelectBySelective(WmNewsPageReqDto dto, Long uid);

}
```

（2）WmNewsMapper.xml

在文件resources/mappers/wemedia/WmNewsMapper.xml中实现对应的SQL编写

```
<mapper namespace="com.heima.model.mappers.wemedia.WmNewsMapper">
    <select id="selectBySelective"
    resultType="com.heima.model.media.pojos.WmNews">
        select
        <include refid="Base_Column_List"/>
        from wm_news
        <where>
        user_id = #{uid}
        <if test="dto.status != null and dto.status != -1" >
        and status = #{dto.status}
        </if>
        <if test="dto.channelId != null" >
        and channel_id = #{dto.channelId}
        </if>
        <if test="dto.keyWord != null" >
        and title like concat('%', #{dto.keyWord}, '%')
        </if>
        <if test="dto.beginPubdate != null" >
        and publish_time <![CDATA[>]]> #{dto.beginPubdate}
        </if>
        <if test="dto.endPubdate != null">
        and publish_time <![CDATA[<]]> #{dto.endPubdate}
        </if>
        </where>
        limit ${(dto.page - 1) * dto.size}, ${dto.size}
    </select>
    <select id="countSelectBySelective" resultType="java.lang.Integer">
        select
        count(1)
        from wm_news
        <where>
        user_id = #{uid}
        <if test="dto.status != null and dto.status != -1" >
        and status = #{dto.status}
        </if>
        <if test="dto.channelId != null" >
```

```
        and channel_id = #{dto.channelId}
        </if>
        <if test="dto.keyWord != null" >
        and title like concat('%', #{dto.keyWord}, '%')
        </if>
        <if test="dto.beginPubdate != null" >
        and publish_time <![CDATA[>]]> #{dto.beginPubdate}
        </if>
        <if test="dto.endPubdate != null">
        and publish_time <![CDATA[<]]> #{dto.endPubdate}
        </if>
        </where>
    </select>
</mapper>
```

5.3.3 时序说明

- 如果用户传递参数为空返回PARAM_REQUIRE错误
- 检测参数是否合法
- 查询用户相关的图文数据
- 统计当前与用户相关的图文数据一共多少
- 流程处理完成返回处理结果

5.3.4 代码实现

（1）NewsService

在类com.heima.media.service.NewsService中定义listByUser方法

```
public interface NewsService {
    /**
    * 查询发布库中当前用户文章信息
    * @param dto
    * @return
    */
    ResponseResult listByUser(WmNewsPageReqDto dto);
}
```

（2）NewsServiceImpl

在类：com.heima.media.service.impl.NewsServiceImpl中实现接口中的方法，在此处我们实现了listByUser，用于实现查询当前用户的文章

```
@Override
public ResponseResult listByUser(WmNewsPageReqDto dto) {
    if (dto == null) {
        return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_INVALID);
    }
    //检测参数
    dto.checkParam();
    Long uid = WmThreadLocalUtils.getUser().getId();
    List<WmNews> datas = wmNewsMapper.selectBySelective(dto, uid);
    int total = wmNewsMapper.countSelectBySelective(dto, uid);
    PageResponseResult responseResult = new PageResponseResult(dto.getPage(),
                                                               dto.getSize(),
total);
```

```
        responseResult.setData(datas);
        responseResult.setHost(fileServerUrl);
        return responseResult;
    }
```

（3）WmNewsPageReqDto

创建类：com.heima.media.mysql.core.model.dtos.WmNewsPageReqDto，此类在model模块中创建，定义请求入参，实现如下：

```
@Data
public class WmNewsPageReqDto extends PageRequestDto {
    private Short status;
    private Date beginPubdate;
    private Date endPubdate;
    @IdEncrypt
    private Integer channelId;
    private String keyWord;
}
```

（4）NewsControllerApi

在类com.heima.media.apis.NewsControllerApi中增加listByUser接口方法

```
    /**
    * 用户查询
    * @return
    */
    ResponseResult listByUser(WmNewsPageReqDto dto);
```

（5）NewsController

在com.heima.media.controller.v1.NewsController类中实现NewsControllerApi接口方法，调用对应的service接口即可。

```
@RestController
@RequestMapping("/api/v1/media/news")
public class NewsController implements NewsControllerApi {
    @Autowired
    private NewsService newsService;

    @PostMapping("/list")
    @Override
    public ResponseResult listByUser(@RequestBody WmNewsPageReqDto dto) {
        return newsService.listByUser(dto);
    }
}
```

5.4 文章详情后台接口

5.4.1 接口定义

（1）基本定义

文章详情主要用于查看和编辑数据初始化.

参考标准	请参考通用接口规范
发布接口名称	/api/v1/media/news/news
请求DTO	com.heima.model.media.dtos.WmNewsDto
响应DTO	{ "host":, "code": 0, "error_message": "操作成功", "data": {} }

（2）CODE定义

PARAM_INVALID	PARAM_INVALID(501,"无效参数"),

5.4.2 Mapper实现

（1）WmNewsMapper

在类com.heima.model.mappers.wemedia.WmNewsMapper中定义selectNewsDetailByPrimaryKey方法：

```
WmNews selectNewsDetailByPrimaryKey(Integer id);
```

（2）WmNewsMapper.xml

在文件resources/mappers/wemedia/ WmNewsMapper.xml中编写接口对应的SQL语句

```
<select id="selectNewsDetailByPrimaryKey"
        resultType="com.heima.model.media.pojos.WmNews">
    select <include refid="Base_Column_List"/>
    from wm_news
    where id = #{id}
</select>
```

5.4.3 时序说明

- 如果用户传递参数为空返回PARAM_REQUIRE错误
- 根据id查询当前文章
- 如果查询不到对应的文章则直接返回错误提示
- 流程处理完成返回处理结果

5.4.4 代码实现

（1）NewsService

在类com.heima.media.service.NewsService中定义方法findWmNewsById

```
/**
     * 根据文章id查询文章
     * @return
     */
ResponseResult findWmNewsById(WmNewsDto wmNews);
```

（2）NewsServiceImpl

在类com.heima.media.service.impl.NewsServiceImpl中实现对应的方法

```
@Override
public ResponseResult findWmNewsById(WmNewsDto dto) {
    if (dto == null || dto.getId() == null) {
        return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_REQUIRE, "文章ID不可缺少");
    }
    WmNews wmNews = wmNewsMapper.selectNewsDetailByPrimaryKey(dto.getId());
    if (wmNews == null) {
        return ResponseResult.errorResult(AppHttpCodeEnum.DATA_NOT_EXIST, "文章不存在");
    }
    ResponseResult responseResult = ResponseResult.okResult(wmNews);
    responseResult.setHost(fileServerUrl);
    return responseResult;
}
```

（3）NewsControllerApi

在类com.heima.media.apis.NewsControllerApi中增加summitNews、saveDraftNews方法

```
/**
    * 根据id获取文章信息
    * @param id
    * @return
    */
ResponseResult wmNews(@RequestBody WmNewsDto wmNews);
```

（4）NewsController

在com.heima.media.controller.v1.NewsController类中实现NewsControllerApi接口方法，调用对应的service接口即可。

```
@PostMapping("/news")
@Override
public ResponseResult wmNews(@RequestBody WmNewsDto dto) {
    return newsService.findWmNewsById(dto);
}
```

5.5 删除文章后台接口

5.5.1 接口定义

（1）基本定义

自从对于未发布的文章进行删除操作.

参考标准	请参考通用接口规范
发布接口名称	/api/v1/media/news/del_news
请求DTO	com.heima.model.media.dtos.WmNewsDto
响应DTO	{ "host":, "code": 0, "error_message": "操作成功", "data": {} }

（2）CODE定义

PARAM_INVALID	PARAM_INVALID(501,"无效参数"),

5.5.2 Mapper实现

（1）WmNewsMapper

在类com.heima.model.mappers.wemedia.WmNewsMapper中定义以下方法

```
    WmNews selectByPrimaryKey(Integer id);
    int deleteByPrimaryKey(Integer id);
```

（2）WmNewsMapper.xml

在文件resources/mappers/wemedia/ WmNewsMapper.xml中增加对应SQL实现

```
<delete id="deleteByPrimaryKey" parameterType="java.lang.Integer" >
    delete from wm_news
    where id = #{id}
  </delete>

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

5.5.3 时序说明

- 如果用户传递参数为空返回PARAM_REQUIRE错误
- 根据id查询当前文章
- 如果查询不到对应的文章则直接返回错误提示
- 流程处理完成返回处理结果

5.5.4 代码实现

（1）NewsService

在类com.heima.media.service.NewsService：中定义delNews方法

```
/**
    **
    * @param id
    * @return
    */
ResponseResult delNews(WmNewsDto wmNews);
```

（2）NewsServiceImpl

在类com.heima.media.service.impl.NewsServiceImpl中实现接口中方法，此处需要注意不仅仅需要删除文章数据还需要删除文章资源关联数据

```
@Override
public ResponseResult delNews(WmNewsDto dto) {
    if (dto == null || dto.getId() == null) {
        return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_INVALID);
    }
    WmNews wmNews = wmNewsMapper.selectByPrimaryKey(dto.getId());
    if (wmNews == null) {
        return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_INVALID, "文章不存在");
    }
    //判断是否审核通过
    if (WmMediaConstans.WM_NEWS_AUTHED_STATUS.equals(wmNews.getStatus()) ||
        WmMediaConstans.WM_NEWS_PUBLISH_STATUS.equals(wmNews.getStatus())) {
        return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_INVALID, "当前文章已通过审核不可删除");
    }
    //删除文章素材关联表信息
    wmNewsMaterialMapper.delByNewsId(wmNews.getId());
    //删除文章信息
    wmNewsMapper.deleteByPrimaryKey(wmNews.getId());
    return ResponseResult.okResult(AppHttpCodeEnum.SUCCESS);
}
```

（3）NewsControllerApi

在类com.heima.media.apis.NewsControllerApi中增加delNews方法

```
    /**
    * 删除文章
    * @param id
    * @return
    */
    ResponseResult delNews(@RequestBody WmNewsDto wmNews);
```

（4）NewsController

在com.heima.media.controller.v1.NewsController类中实现NewsControllerApi接口方法，调用对应的service接口即可。

```
@PostMapping("/del_news")
@Override
public ResponseResult delNews(@RequestBody WmNewsDto dto) {
    return newsService.delNews(dto);
}
```

5.6 文章内容列表-前台

在内容列表界面中我们主要实现了对文章的检索功能。

5.6.1 接口定义

（1）删除文章

- 在src/constants/api.js中定义常量映射到后端请求地址

```
export const API_ARTICLES_DELETE = '/api/v1/media/news/del_news' //删除文章
```

- 在src/api/content.js中定义请求方法（此处省略了引入刚才定义的常量，此后所有导入省略请自行导入需要的常量及方法）

```
export function deleteArticles (articlesId) {
    return Request({
        url:API_ARTICLES_DELETE ,
        method:'post',
        params:{},
        data:{id:articlesId}
    })
}
```

（2）检索文章

- 在src/constants/api.js中定义常量映射到后端请求地址

```
export const API_SEARCHARTICELS = '/api/v1/media/news/list' //检索文章
```

- 在src/api/content.js中定义请求方法

```
//搜索文章
export function searchArticle (data) {
    return Request({
        url:API_SEARCHARTICELS,
        method:'post',
        data,
        params:{}
    })
}
```

5.6.2 路由调整

在src/router.js中asyncRouterMap对象的children数组中增加以下改动，以满足全局自动记录路由的功能：

```
{
    path:'/article/list',
    component: () => import('./views/content/index.vue'),
}
```

5.6.3 实现文章列表

（1）搜索工具组件定义

在src/views/content/components/中定义SearchTool.vue组件

```
<template>
  <section class="filter">
    <header>全部图文</header>
    <div class="filter-container">
```

```
      <el-form ref="form">
          <el-form-item label="文章状态：" label-width="110px">
            <a
            v-for='item in stateList'
            :key="item.value"
            href="javascript:;"
            :class="['state_label',(item.value === selectState.value) ? 'active'
: '']"
            @click="changeState(item)">{{item.label}}</a>
          </el-form-item>
        <el-form-item label="频道列表："  label-width="110px">
            <el-select  v-model="channel_id" @change="queryData">
              <el-option
                v-for="item in channel_list"
                :key="item.id"
                :label="item.name"
                :value="item.id">
              </el-option>
            </el-select>
        </el-form-item>
         <el-form-item label="时间选择：  " label-width="110px" >
           <el-date-picker
                type="datetimerange"
                v-model="date"
                range-separator="-"
                start-placeholder="开始日期"
                end-placeholder="结束日期"
                format="yyyy-MM-dd"
                value-format="yyyy-MM-dd"
                placeholder="选择日期"
                @change="queryData"
              />
        </el-form-item>
      </el-form>
    </div>
  </section>
</template>
<script>

export default {
  props: ['changePage','channel_list'],
  data() {
    return {
      stateList:[
        {label:'全部',value:5},
        {label:'草稿',value:0},
        {label:'待审核',value:1},
        {label:'审核通过',value:2},
        {label:'审核失败',value:3},
      ],
      selectState:{
        //选择的筛选状态
        label:'全部',value:5
      },
      channel_id:null, //频道id
      date:null,

    }
```

```
  },
  methods: {
     //查询数据 值得注意的是 一旦条件形成 那么页码应该重新设置为1
    // 因为查询条件的变化  页码应该从第一页开始
    queryData () {
      let params = {
         resetPage:true, //用于判断是否需要重新设置分页器的标记
         channel_id:this.channel_id,
         status: this.selectState.value == 5 ? null : this.selectState.value,
         page:1,
         begin_pubdate:(this.date && this.date.length) ? this.date[0] : null,
         end_pubdate:(this.date && this.date.length > 1) ? this.date[1] : null
      }
     this.changePage && this.changePage(params) //调用上层组件的查询方法
    },
    //切换文章状态
    changeState (state) {
        this.selectState = state //设置状态
        this.queryData() //查询数据
    }
  }
}
</script>
<style rel="stylesheet/scss" lang="scss" scoped>
  .filter {
    background-color: #ffffff;
    text-align: left;
    border: 1px solid #e7e7e9;
    header {
      border-bottom: 1px dashed #cccccc;
      margin: 0 5px;
      padding: 0 10px;
      font-size: 14px;
      height: 55px;
      line-height: 55px;
      color: #323745;
    }
    .filter-container {
      overflow: hidden;
      .el-form {
        padding: 20px 20px 0;
        overflow: hidden;
        .el-form-item {
          margin: 20px 0
        }
      }
      .date-filter {
        padding: 25px 20px 20px;
        overflow: hidden;
        span {
          font-size: 14px;
          margin-right: 20px;
          height: 40px;
          line-height: 40px;
          float: left;
          cursor: pointer;
          &.active, &:hover {
            color: #3296fa;
```

```
          }
        }
        .time-container {
          float: left;
          position: relative;
        }
      }
    }
  }
  .state_label {
    float:left;
    padding-right:25px;
    font-size:14px;
    color:#333;
  }
   .active{
    color:#3296fa
  }
</style>
```

（2）搜索结果组件定义

在src/views/content/components/中定义SearchResult.vue

```
<template>
  <section class="result">
    <header>{{`共找到${total}条符合条件的内容`}}</header>
    <ul class="result-container">
      <li v-for="(item,index) in  articleList" :key='index' class='articles-
item'>
         <img class="article-img" :src="getImage(item)"/>
         <dl class="article-content">
          <dt>
            <a @click="noAction" href="#" class="">{{item.title}}</a>
            <div  @click="operateBtn(item.id,$event)">
              <i data-type='up'  v-if="item.status == '9'&&item.enable=='0'"
class="el-icon-upload2">上架</i>
              <i data-type='down'  v-if="item.status == '9'&&item.enable=='1'"
class="el-icon-download">下架</i>
              <i data-type='modify'  v-if="item.status != '9'" class="el-icon-
edit">修改</i>
              <i v-if="item.status != '9'" data-type='del' class="el-icon-
delete">删除</i>
            </div>
          </dt>
          <dd>
            <el-tag class="draft" v-if="item.status == '0'">草稿</el-tag>
            <el-tag class="audit" v-if="item.status == '1'">待审核</el-tag>
            <el-tag class="audit" v-if="item.status == '3'">待人工审核</el-tag>
            <el-tag class="audit" v-if="item.status == '4'">待发布</el-tag>
            <el-tag class="publish" v-if="item.status == '8'">待发布</el-tag>
            <el-tag class="publish" v-if="item.status == '9'">已发表</el-tag>
            <el-tag class="unaudit" v-if="item.status == '2'">未通过审核:
{item.reason}</el-tag>
            <el-tag class="delete" v-if="item.status == '100'">已删除</el-tag>
            <template v-if="item.status == '9'">
            <el-tag class="draft" v-if="item.enable == '0'">下架</el-tag>
```

```
            <el-tag class="audit" v-if="item.enable == '1'">上架</el-tag>
            </template>
          </dd>
          <dd class="time">{{dateFormat(item.publish_time)}}</dd>
        </dl>
      </li>
    </ul>
    <div class="pagination">
      <el-pagination
        layout="total,prev, pager, next"
        @current-change='pageChange'
        :current-page.sync='listPage.currentPage'
        :page-size="pageSize"
        :total="total">
      </el-pagination>
    </div>
  </section>
</template>

<script>
import DateUtil from '@/utils/date'
const avatar = require('@/assets/avatar.jpg')
export default {
  props:
['host','articleList','pageSize','total','changePage','deleteArticlesById','upOr
Down'],
  data() {
    return {
       listPage:{
          currentPage:1
       }
    }
  },
  methods: {
    noAction : function(){
      alert('该功能暂未实现');
    },
    getImage : function(item){
      if(item.images){
        let temp = item.images.split(",")
        if(temp.length>0){
          return this.host+temp[0];
        }
      }
      return avatar
    },
    //页码变化 调用上层组件的方法
    pageChange (newPage) {
        this.changePage && this.changePage({page:newPage})
    },
    resetPage(){

    },
    //重新设置页码
    dateFormat (time) {
      return DateUtil.format13HH(time)
    },
    //操作
```

```
    operateBtn (Id,event) {
      switch(event.target.dataset.type){
        case 'modify':
        this.$router.push({path:'/article/publish',query:{articleId:Id}})
        break;
        case 'down':
          this.upOrDown(Id,0)
          break;
        case 'up':
          this.upOrDown(Id,1)
          break;
        case 'del':
          this.$confirm('此操作将永久删除该文章, 是否继续?', '提示', {
          confirmButtonText: '确定',
          cancelButtonText: '取消',
          type: 'warning'
        }).then(() => {
           this.deleteArticlesById && this.deleteArticlesById(Id) //删除文件
        }).catch(() => {
          this.$message({
            type: 'info',
            message: '已取消删除'
          });
        });
        break;
      default :

      }
    }
  }
}
</script>
<style rel="stylesheet/scss" lang="scss" scoped>
  .result {
    background-color: #ffffff;
    text-align: left;
    border: 1px solid #e7e7e9;
    margin-top: 10px;
    header {
      border-bottom: 1px dashed #cccccc;
      margin: 0 5px;
      padding: 0 10px;
      font-size: 14px;
      height: 55px;
      line-height: 55px;
      color: #323745;
    }
    li {
      overflow: hidden;
      padding: 15px 5px;
      margin: 0 10px;
      border-bottom: 1px solid #f2f3f5;
       .draft {
           color:#FDC2A9;
           border-color: #FDC2A9;
           background: #FFFFFF;
        }
        .delete {
```

```
            color:rgb(243, 24, 8);
            border-color: rgb(243, 24, 8);
            background: #FFFFFF;
        }
        .audit {
            color:rgb(53, 160, 17);
            border-color: rgb(53, 160, 17);
            background: #FFFFFF;
        }
        .unaudit {
            color:#FFAEAE;
            border:none;
        }
    }
    .article-img {
      float: left;
      width: 150px;
      height: 100px;
    }
    .article-content {
      margin-left: 160px;
      text-align: left;
      dt {
        color: #333333;
        font-size: 14px;
        margin: 0 0 7px;
        div {
          float: right;
          i {
            margin-left: 20px;
            font-size: 12px;
            cursor: pointer;
          }
        }
      }
      .time {
        margin-bottom: 10px;
        span {
          color:#3296fa;
          margin-right: 8px;
        }
      }
      dd {
        font-size: 12px;
        margin: 5px 0;
        color: #999999;
        b {
          color: #333333;
        }
        span {
          margin-right: 10px;
        }
      }
    }
  }
  .el-pagination {
    text-align: center;
    margin: 20px 0 40px;
```

```
  }
</style>
```

（3）日期处理工具

在src/utils/中定义date.js日期处理工具

```
function FormatDate(){}
FormatDate.prototype= {
    formatDate:function(date, fmt) {
        if (/(y+)/.test(fmt)) {
            fmt = fmt.replace(RegExp.$1, (date.getFullYear() + '').substr(4 -
RegExp.$1.length))
        }
        let o = {
            'M+': date.getMonth() + 1,
            'd+': date.getDate(),
            'h+': date.getHours(),
            'm+': date.getMinutes(),
            's+': date.getSeconds()
        }
        for (let k in o) {
            if (new RegExp(`(${k})`).test(fmt)) {
                let str = o[k] + ''
                fmt = fmt.replace(RegExp.$1, RegExp.$1.length === 1 ? str :
this.padLeftZero(str))
            }
        }
        return fmt
    },
    padLeftZero:function (str) {
        return ('00' + str).substr(str.length)
    },
    format10:function(time){
        return this.format13(time*1000);
    },
    format13:function(time){
      if(time==undefined){
        return ""
      }
        let date = new Date(time);
        return this.formatDate(date,'yyyy-MM-dd')
    },
    format13HH:function(time){
      if(time==undefined){
        return ""
      }
      let date = new Date(time);
      return this.formatDate(date,'yyyy-MM-dd hh:mm:ss')
    },
    // 最近几天时间
    getNearTime:function(AddDayCount) {
      var dd = new Date();
      return dd.getTime()-AddDayCount*24*3600000;
    },
    // 最近本周开始时间
    getWeekSTime:function() {
```

```
      var dd = new Date();
      dd.setDate(dd.getDate() -dd.getDay());
      return dd.getTime();
    },
    // 最近本周结束时间
    getWeekETime:function() {
      var dd = new Date();
      dd.setDate(dd.getDate() +(7-dd.getDay()));
      return dd.getTime();
    },
    diffTime:function(time){
        if(time.length==10){
            time = parseInt(time)*1000;
        }
        var nowDate = new Date().getTime(),
            oldDate = new Date(time).getTime(),
            diffTime = parseInt((nowDate - oldDate)/1000,10),
            oneMinute = 60,
            oneHour = 60 * oneMinute,
            oneDay = 24 * oneHour,
            oneMonth = 30 * oneDay,
            oneYear = 12 * oneMonth,
            compareArr = [oneYear,oneMonth,oneDay,oneHour,oneMinute],
            postfix = ['年前','个月前','天前','个小时前','分钟前','1分钟内'],
            diffYear,diffMonth,diffDay,diffHour,diffMinute,len=5;
        for(var i =0; i< len ;i++){
            var diff = Math.floor(diffTime/compareArr[i]);
            if(diff > 0){
                return diff + postfix[i];
            }
            else if(i === len -1 && diff === 0){
                return postfix[len];
            }
        }
    }
}
export  default new FormatDate()
```

（4）文章列表实现

在src/views/content/中定义index.vue，具体代码如下：

```
  <template>
    <div class="">
      <search-tool
        v-if="!searchText"
        :changePage="searchArticle"
        :channel_list="channel_list"
      />
      <search-result
        ref='mySearchResult'
        :articleList="articleList"
        :host="host"
        :total="total"
        :changePage="searchArticle"
        :pageSize="params.size"
        :deleteArticlesById="deleteArticlesById"
```

```
        :upOrDown="upOrDown"
      />
    </div>
  </template>

  <script>
    import SearchTool from './components/SearchTool.vue'
    import SearchResult from './components/SearchResult.vue'
    import { deleteArticles , searchArticle,upDownArticle} from  '@/api/content'
    import { getChannels } from  '@/api/publish'
    export default {
      name: 'ContentManage',
      data() {
        return {
          channel_list:[],
          articleList:[],
          total:0,
          host:'',
          searchText:null,
          params:{
            page:1,
            size:10
          }, //查询参数  用于全局存储 因为分页时 需要在查询条件基础上分页
          tempParams : {}
        }
      },
      created () {
        let { searchText } = this.$route.query //从路由中查找关键字参数
        this.searchText = searchText //存储当前值
        this.getChannels() //拉取频道列表数据
        // 如果搜索关键字有值 则直接调用搜索接口 否则 调用默认接口
        this.searchArticle();
      },
      components: {
        SearchTool,
        SearchResult
      },
      computed: {

      },
      methods: {
        //搜索文章
        async searchArticle (newParams) {
          this.tempParams = newParams
          let result = await
searchArticle({...this.params,key_word:this.searchText,...this.tempParams})
          /****需要重新将分页器的页码设置为1******/
          if(this.$refs.mySearchResult){
            this.$refs.mySearchResult.resetPage(); //重置
          }
          this.host = result.host
          this.total = result.total //总记录数
          this.articleList = result.data //当前的数组
        },
        //根据Id删除文章
        async deleteArticlesById (Id) {
          let temp = await deleteArticles(Id)
          if(temp.code==0) {
```

```
            this.$message({type: 'success', message: '删除成功!'});
            this.searchArticle();
          }else{
            this.$message({type: 'error', message: temp.error_message});
          }
        },
        //上下架
        async upOrDown (Id,enable) {
          let temp = await upDownArticle({id:Id,enable:enable})
          if(temp.code==0) {
            this.$message({type: 'success', message: '操作成功!'});
            this.searchArticle();
          }else{
            this.$message({type: 'error', message: temp.error_message});
          }
        },
        //拉取频道数据
        async getChannels () {
          let result = await getChannels()
          this.channel_list = result.data  //赋值数据
        }
      }
    }
  </script>
```

6 图文和粉丝统计报表

6.1 图文统计后台接口

6.1.1 接口定义

（1）基本定义

图文统计涉及时间等查询条件，因此也需要请求DTO。

参考标准	请参考通用接口规范
发布接口名称	/api/v1/statistics/news
请求DTO	com.heima.model.media.dtos.StatisticDto
响应DTO	{"host":, "code": 0, "error_message": "操作成功","data": {} }

（2）CODE定义

PARAM_INVALID	PARAM_INVALID(501,"无效参数"),

6.1.2 Mapper实现

（1）WmNewsStatisticsMapper

创建类com.heima.model.mappers.wemedia.WmNewsStatisticsMapper：并定义findByTimeAndUserId根据时间和用户ID查询相关数据

```
public interface WmNewsStatisticsMapper {
    List<WmNewsStatistics> findByTimeAndUserId(String burst, Long userId,
    StatisticDto dto);
}
```

WmNewsStatisticsMapper.xml

创建文件resources/mappers/wemedia/ WmNewsStatisticsMapper

```
<mapper namespace="com.heima.model.mappers.wemedia.WmNewsStatisticsMapper">
    <resultMap id="BaseResultMap"
    type="com.heima.model.media.pojos.WmNewsStatistics" >
        <id column="id" property="id"/>
        <result column="user_id" property="userId" />
        <result column="article" property="article" />
        <result column="read_count" property="readCount" />
        <result column="comment" property="comment" />
        <result column="follow" property="follow" />
        <result column="collection" property="collection" />
        <result column="forward" property="forward" />
        <result column="likes" property="likes" />
        <result column="unlikes" property="unlikes" />
        <result column="unfollow" property="unfollow" />
        <result column="created_time" property="createdTime" />
    </resultMap>
    <sql id="Base_Column_List" >
        id, user_id, article, read_count, comment, follow, collection, forward,
        likes, unlikes,
        unfollow, created_time
    </sql>

    <select id="findByTimeAndUserId"
    resultType="com.heima.model.media.pojos.WmNewsStatistics">
        /*!mycat:sql=select id from wm_news_statistics where burst='${burst}'*/

        select <include refid="Base_Column_List"/>
        from wm_news_statistics
        <where>
        user_id = #{userId}
        <if test="dto.type == 0">
        and date(created_time) = CURDATE()
        </if>
        <if test="dto.type != 0 and dto.stime != null">
        and date(created_time) <![CDATA[ >= ]]> date(#{dto.stime})
        </if>
        <if test="dto.type != 0 and dto.etime != null">
        and date(created_time) <![CDATA[ <= ]]> date(#{dto.etime})
        </if>
        </where>
    </select>
</mapper>
```

(2)根据id查询用户

在WmUserMapper接口新增方法

```
 WmUser selectById(Long id);
```

WmUserMapper.xml

```
 <!-- 通过id查询用户 -->
<select id="selectById" resultMap="BaseResultMap">
    select
    <include refid="Base_Column_List" />
    from wm_user
    where id = #{id}
</select>
```

6.1.3 时序说明

- 如果用户传递参数为空返回PARAM_REQUIRE错误
- 查询当前用户信息
- 根据条件查询相应的数据
- 流程处理完成返回处理结果

6.1.5 代码实现

（1）StatisticsService

创建类：com.heima.media.service.StatisticsService：

```
public interface StatisticsService {
    /**
    * 查找图文统计数据
    * @param dto
    * @return
    */
    ResponseResult findWmNewsStatistics(StatisticDto dto);
}
```

（2）StatisticsServiceImpl

创建类：com.heima.media.service.impl.StatisticsServiceImpl

```
@Service
@SuppressWarnings("all")
public class StatisticsServiceImpl implements StatisticsService {
    @Autowired
    private WmNewsStatisticsMapper wmNewsStatisticsMapper;

    @Autowired
    private WmUserMapper wmUserMapper;

    @Override
    public ResponseResult findWmNewsStatistics(StatisticDto dto) {
        ResponseResult responseResult = check(dto);
        if (responseResult != null){
            return responseResult;
        }
```

```
        WmUser wmUser = queryAllUserInfo();
        String burst = BurstUtils.groudOne(wmUser.getApUserId());
        return

 ResponseResult.okResult(wmNewsStatisticsMapper.findByTimeAndUserId(burst,
                        wmUser.getApUserId(), dto));
    }

    private WmUser queryAllUserInfo() {
        WmUser user = WmThreadLocalUtils.getUser();
        user = wmUserMapper.selectById(user.getId());
        return user;
    }

    private ResponseResult check(StatisticDto dto) {
        if (dto == null && dto.getType() == null) {
            return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_INVALID);
        }
        if (WmMediaConstans.WM_NEWS_STATISTIC_CUR != dto.getType() &&
                (dto.getStime() == null || dto.getEtime() == null)) {
            return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_INVALID);
        }
        return null;
    }
}
```

（3）StatisticDto

创建类：com.heima.media.mysql.core.model.dtos.StatisticDto

此类在model模块中创建，定义请求入参，实现如下：

```
@Data
public class StatisticDto {
private Short type;
private Date stime;
private Date etime;
private List<String> time;
}
```

（4）StatisticsControllerApi

在类com.heima.media.apis.StatisticsControllerApi中增加方法

```
public interface StatisticsControllerApi {
    /**
    * 文章数据
    * @param dto
    * @return
    */
    public ResponseResult newsData(StatisticDto dto);
}
```

（5）StatisticsController

在com.heima.media.controller.v1.StatisticsController类中实现接口方法，调用对应的service接口即可。

```
@RestController
@RequestMapping("/api/v1/statistics")
public class StatisticsController implements StatisticsControllerApi {
    @Autowired
    private StatisticsService statisticsService;

    @Override
    @RequestMapping("/news")
    public ResponseResult newsData(@RequestBody StatisticDto dto) {
        return statisticsService.findWmNewsStatistics(dto);
    }
}
```

6.2 粉丝统计后台接口

6.2.1 接口定义

（1）基本定义

粉丝统计与

参考标准	请参考通用接口规范
发布接口名称	/api/v1/statistics/fans
请求DTO	com.heima.model.media.dtos.StatisticDto
响应DTO	{ "host":, "code": 0, "error_message": "操作成功", "data": {} }

（2）CODE定义

PARAM_INVALID	PARAM_INVALID(501,"无效参数"),

6.2.2 Mapper实现

（1）WmFansStatisticsMapper

创建类com.heima.model.mappers.wemedia.WmFansStatisticsMapper：

```
public interface WmFansStatisticsMapper {
    List<WmFansStatistics> findByTimeAndUserId(String burst, Long userId,
    StatisticDto dto);
}
```

（2）WmFansStatisticsMapper.xml

创建文件resources/mappers/wemedia/WmFansStatisticsMapper.xml

```
<mapper namespace="com.heima.model.mappers.wemedia.WmFansStatisticsMapper">
    <resultMap id="BaseResultMap"
type="com.heima.model.media.pojos.WmFansStatistics" >
        <id column="id" property="id" />
        <result column="user_id" property="userId" />
```

```
        <result column="article" property="article" />
        <result column="read_count" property="readCount" />
        <result column="comment" property="comment" />
        <result column="follow" property="follow" />
        <result column="collection" property="collection" />
        <result column="forward" property="forward" />
        <result column="likes" property="likes" />
        <result column="unlikes" property="unlikes" />
        <result column="unfollow" property="unfollow" />
        <result column="created_time" property="createdTime" />
        <result column="burst" property="burst" />
    </resultMap>
    <sql id="Base_Column_List" >
    id, user_id, article, read_count, comment, follow, collection, forward,
likes, unlikes,
    unfollow, created_time
  </sql>
    <select id="findByTimeAndUserId"
            resultType="com.heima.model.media.pojos.WmFansStatistics">
        /*!mycat:sql=select id from wm_fans_statistics where burst='${burst}'*/
        select <include refid="Base_Column_List"/>
        from wm_fans_statistics
        <where>
            user_id = #{userId}
            <if test="dto.type == 0">
                and date(created_time) = CURDATE()
            </if>
            <if test="dto.type != 0 and dto.stime != null">
                and date(created_time) <![CDATA[ >= ]]> date(#{dto.stime})
            </if>
            <if test="dto.type != 0 and dto.etime != null">
                and date(created_time) <![CDATA[ <= ]]> date(#{dto.etime})
            </if>
        </where>
    </select>
</mapper>
```

6.2.3 时序说明

- 如果用户传递参数为空返回PARAM_REQUIRE错误
- 查询当前用户信息
- 根据条件查询相应的数据
- 流程处理完成返回处理结果

6.2.4 代码实现

（1）StatisticsService

在com.heima.media.service.StatisticsService中增加粉丝数据接口方法

```
/**
    * 用户粉丝统计数据
    * @param dto
    * @return
    */
ResponseResult findFansStatistics(StatisticDto dto);
```

（2）StatisticsServiceImpl

在com.heima.media.service.impl.StatisticsServiceImpl类中实现接口方法

```
@Autowired
private WmFansStatisticsMapper wmFansStatisticsMapper;

@Override
public ResponseResult findFansStatistics(StatisticDto dto) {
    ResponseResult responseResult = check(dto);
    if (responseResult != null){
        return responseResult;
    }
    WmUser wmUser = queryAllUserInfo();
    Long userId = wmUser.getApUserId();
    String burst = BurstUtils.groudOne(userId);
    List<WmFansStatistics> datas =
        wmFansStatisticsMapper.findByTimeAndUserId(burst, userId, dto);
    return ResponseResult.okResult(datas);
}
```

（3）StatisticDto

创建类：com.heima.media.mysql.core.model.dtos.StatisticDto

此类在model模块中创建，定义请求入参，实现如下：

```
@Data
public class StatisticDto {
    private Short type;
    private Date stime;
    private Date etime;
    private List<String> time;
}
```

（4）StatisticsControllerApi

在类com.heima.media.apis.StatisticsControllerApi中增加方法

```
/**
* 粉丝数据*
* @param dto*
* @return*
*/
public ResponseResult fansData(@RequestBody StatisticDto dto);
```

（5）StatisticsController

在com.heima.media.controller.v1.StatisticsController类中实现接口方法，调用对应的service接口即可。

```
@Override
@RequestMapping("/fans")
public ResponseResult fansData(@RequestBody StatisticDto dto) {
    return statisticsService.findFansStatistics(dto);
}
```

6.3 图文数据前台开发

图文数据界面中我们主要实现了当前用户的图文数据的统计功能，并以图表的形式进行了展示。在这里的相关页面需要使用到echarts，需要在项目中安装echarts，后页面中导入使用。

6.3.1 接口定义

（1）查询图文数据

- 在src/constants/api.js中定义常量映射到后端请求地址

```
export const API_STATISTICS_NEWS = '/api/v1/statistics/news' //图文统计
```

- 在src/api/content.js中定义请求方法（此处省略了引入刚才定义的常量，此后所有导入省略请自行导入需要的常量及方法）

```
//获取统计数据
export function getNewsStatistics(data) {
    return Request({
        url:API_STATISTICS_NEWS,
        method:'post',
        params:{},
        data:data
    })
}
```

（2）查询粉丝数据

- 在src/constants/api.js中定义常量映射到后端请求地址

```
export const  API_GET_FANS_STATISTIC = '/api/v1/statistics/fans' //粉丝统计数据
```

- 在src/api/fans.js中定义请求方法

```
//粉丝数据
export function getFansStatistics(data) {
  return Request({
    url: API_GET_FANS_STATISTIC,
    method: 'post',
    data
  })
}
```

5.4.2 路由调整

在src/router.js中asyncRouterMap对象的children数组中增加以下改动，以满足全局自动记录路由的功能：

```
    {
        path:'/material/data',
        component: () => import('./views/content/detail.vue'),
    },
    {
        path:'/fans/index',
        component: () => import('./views/fans/index.vue'),
      }
```

5.4.3 实现图文数据

（1）统计组件定义

在src/views/content/components/中定义Statist.vue， 实现基本数据的展示，具体代码如下：

```
<template>
  <div class="content">
    <el-row :gutter="40">
      <el-col :span="8">
        <div class="grid-content">
          <i></i>
          <div>
            <div>{{article}} 个</div>
            <span>图文发布量</span>
          </div>
        </div>
      </el-col>
      <el-col :span="8">
        <div class="grid-content">
          <i></i>
          <div>
            <div>{{likes}} 个</div>
            <span>点赞数量</span>
          </div>
        </div>
      </el-col>
      <el-col :span="8">
        <div class="grid-content">
          <i></i>
          <div>
            <div>{{collection}}</div>
            <span>收藏量</span>
          </div>
        </div>
      </el-col>
    </el-row>
  </div>
</template>

<script>
export default {
  props:["collection","likes","article"],
  components: {
  },
  computed: {

  },
```

```
  methods: {

  }
}
</script>

<style rel="stylesheet/scss" lang="scss" scoped>
.content {
  padding: 0 20px 20px;
}
.el-row {
  .el-col {
    .grid-content {
      background-color: #f8f8f8;
      padding: 30px 0;
      display: flex;
      justify-content: center;
      i {
        width: 50px;
        height: 50px;
        background-color: red;
        border-radius: 50%;
      }
      & > div {
        margin: 0px 0 0 10px;
        span {
          color: #444444;
          font-size: 14px;
        }
        div {
          font-size: 24px;
        }
      }
    }
  }
}

</style>
```

（2）线形图组件定义

在src/views/content/components/中定义LineChart.vue实现数据的线图展示功能：

```
<template>
  <div class="chart-content">
    <div ref="chart" :style="{height:height,width:width}"/>
  </div>
</template>

<script>
import echarts from 'echarts'
require('echarts/theme/macarons') // echarts theme

export default {
  props: {
    className: {
      type: String,
```

```
      default: 'chart'
    },
    width: {
      type: String,
      default: '100%'
    },
    height: {
      type: String,
      default: '350px'
    },
    autoResize: {
      type: Boolean,
      default: true
    },
    chartData: {
      type: Object,
      required: false
    }
  },
  data() {
    return {
      chart: null,
      sidebarElm: null
    }
  },
  watch: {
    chartData: {
      deep: true,
      handler(val) {
        this.setOptions(val)
      }
    }
  },
  mounted() {
    this.initChart()
    // 监听侧边栏的变化
    this.sidebarElm = document.getElementsByClassName('sidebar-container')[0]
    this.sidebarElm && this.sidebarElm.addEventListener('transitionend',
this.sidebarResizeHandler)
  },
  beforeDestroy() {
    if (!this.chart) {
      return
    }
    if (this.autoResize) {
      window.removeEventListener('resize', this.__resizeHandler)
    }

    this.sidebarElm && this.sidebarElm.removeEventListener('transitionend',
this.sidebarResizeHandler)

    this.chart.dispose()
    this.chart = null
  },
  methods: {
    sidebarResizeHandler(e) {
      if (e.propertyName === 'width') {
        this.__resizeHandler()
```

```
      }
    },
    setOptions(lineOption) {
      lineOption = lineOption?lineOption:{
        xAxis: {
          type: 'category',
          boundaryGap: false,
          data: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
        },
        yAxis: {
          type: 'value'
        },
        series: [{
          data: [820, 932, 901, 934, 1290, 1330, 1320],
          type: 'line',
          areaStyle: {}
        }]
      }
      this.chart.setOption(lineOption)
    },
    initChart() {
      this.chart = echarts.init(this.$refs.chart, 'macarons')
      this.setOptions()
    }
  }
}
</script>

<style rel="stylesheet/scss" lang="scss" scoped>
  .chart-content {
    padding: 0 30px;
  }
</style>
```

（3）doughnut图表组件定义

在src/views/fans/components/中定义组件DoughnutChart.vue

```
<template>
  <section class="chart">
    <div class="box">
      <div ref="chart" :style="{height:height,width:width}" class="doughnut"/>
    </div>
  </section>
</template>

<script>
import echarts from "echarts";
require("echarts/theme/macarons"); // echarts theme
export default {
  props: {
    width: {
      type: String,
      default: "100%"
    },
    height: {
      type: String,
```

```
      default: "380px"
    },
    data : {
      type:Object
    }
  },
  data() {
    return {
      chart: null
    };
  },
  watch:{
    data:function(val){
      this.initChart()
    }
  },
  mounted() {
    this.initChart();
  },
  beforeDestroy() {
    if (!this.chart) {
      return;
    }
    this.chart.dispose();
    this.chart = null;
  },
  methods: {
    initChart() {
      this.chart = echarts.init(this.$refs.chart, "macarons");
      let temp = this.data.legend
      if(temp.length>13){
        temp = temp.splice(0,14)
      }
      this.chart.setOption({
        title: {text: this.data.title},
        tooltip: {trigger: "item",formatter: "<br/>{b}: {c} ({d}%)"},
        legend: {y: 'bottom',data:temp},
        series: [
          {
            name: this.data.title,
            type: "pie",
            radius: ["50%","70%"],
            roseType: 'radius',
            avoidLabelOverlap: true,
            itemStyle:{
              color: function(){return "#"+("00000"+
((Math.random()*16777215+0.5)>>0).toString(16)).slice(-6); }
            },
            label: {
              normal: {show: false,position: "center"},
              emphasis: {show: true,textStyle: {fontSize: "30",fontWeight:
"bold"}}
            },
            labelLine: {normal: {show: false}},
            data: this.data.data
          }
        ]
      });
```

```
    }
  }
};
</script>
<style rel="stylesheet/scss" lang="scss" scoped>
.chart {
  margin-top: 10px;
  width: 50%;
  float: left;
  font-size: 14px;
  .box {
    background-color: #fbfbfb;
    overflow: hidden;
    .doughnut {
      float: left;
    }
  }
  .legend {
    margin: 0 10px 10px 10px;
    margin-left: 60%;
  }
}
</style>
```

（4）数据统计整体实现

在src/views/content/中定义组件detail.vue

```
<template>
  <div class="detail-container">
    <header>详情分析</header>
    <div class="filter">
      <el-radio-group  @change="loadDataByButton" v-model="parms.type">
        <el-radio-button  label="0">今日</el-radio-button>
        <el-radio-button  label="1">本周</el-radio-button>
        <el-radio-button  label="7">近7天</el-radio-button>
        <el-radio-button  label="30">近30天</el-radio-button>
      </el-radio-group>
      <el-date-picker v-model="parms.time" type="datetimerange"
                      range-separator="-"
                      start-placeholder="开始日期"
                      end-placeholder="结束日期"  @change="loadDataByTimeRange"
:picker-options="pickerOptions" format="yyyy-MM-dd HH:mm:ss" placeholder="选择日
期"/>
    </div>
    <Statist :article="all.article" :likes="all.likes"
:collection="all.collection"/>
    <line-chart ref="lineChart"/>
    <div class="chart">
      <template v-for="item in pie">
        <doughnut-chart :data="item" v-if="item.title !='发文量-转化率'"/>
      </template>
    </div>
  </div>
</template>

<script>
```

```
import Statist from './components/Statist.vue'
import LineChart from './components/LineChart.vue'
import DoughnutChart from './components/DoughnutChart.vue'
import {getNewsStatistics} from '@/api/content'
import DateUtil from '@/utils/date'
export default {
  name: 'ContentManage',
  data() {
    return {
      parms:{
        type:'0',
        stime:'',
        etime:''
      },
      all:{},
      list:'',
      graph:'',
      pie:{},
      lineInfo : [
        {name:'发文量',type:'article'},
        {name:'阅读量',type:'read_count'},
        {name:'点赞量',type:'likes'},
        {name:'评论量',type:'comment'},
        {name:'收藏量',type:'collection'},
        {name:'转发量',type:'follow'},
        {name:'不喜欢',type:'unlikes'}
      ],
      pickerOptions: {
        disabledDate(time) {
          return time.getTime() > Date.now()
        }
      }
    }
  },
  components: {
    Statist,
    LineChart,
    DoughnutChart
  },
  created(){
    this.getNewsStatistics()
  },
  methods : {
    loadDataByTimeRange:function(e){
      this.parms.type=-1
      this.parms.stime=e[0].getTime()
      this.parms.etime=e[1].getTime()
      this.getNewsStatistics();
    },
    loadDataByButton:function(e){
      if(e=='1'){// 本周
        this.parms.stime=DateUtil.getWeekSTime()
        this.parms.etime=DateUtil.getWeekETime()
      }else{
        this.parms.etime=DateUtil.getNearTime(0)
        this.parms.stime=DateUtil.getNearTime(e)
      }
      this.getNewsStatistics();
```

```
    },
    async getNewsStatistics (){
      let result = await getNewsStatistics(this.parms)
      this.list = result.data
      let all =
{article:0,likes:0,collection:0,forward:0,comment:0,read_count:0}
      let chats = {}
      for (let i = 0; i < result.data.length; i++) {
        let tmp = result.data[i];
        let time = DateUtil.format13(tmp.created_time)
        let data = chats[time]?chats[time]:{}
        for (let j = 0; j <this.lineInfo.length ; j++) {
          let k=this.lineInfo[j].type
          all[k]+=tmp[k]
          let val = data[k]?data[k]:0
          val +=tmp[k]
          data[k]=val
        }
        chats[time]=data
      }
      this.all = all
      this.graph = chats
      this.parseToLine(chats,all)
    },
    parseToLine : function(chats,all){
      // 排序
      var name = [];
      for (let k in chats) {
        name.push(k)
      }
      name.sort()
      let series = {}//折线图数据
      let pie = {}//饼图数据
      for (let i = 0; i <name.length ; i++) {
        for (let j = 0; j <this.lineInfo.length ; j++) {
          let k=this.lineInfo[j].type
          series[k] = series[k]?series[k]:[]
          series[k].push(chats[name[i]][k])
          pie[k] = pie[k]?pie[k]:{}
          pie[k]['title'] = this.lineInfo[j].name+' - 占比'
          pie[k]['data'] = pie[k]['data']?pie[k]['data']:[]
          pie[k]['legend'] = pie[k]['legend']?pie[k]['legend']:[]
          pie[k]['legend'].push(name[i])
          pie[k]['data'].push({value:chats[name[i]][k],name:name[i]})
        }
      }
      let data = []
      let legend=[]
      for (let i = 0; i <this.lineInfo.length ; i++) {
        data.push({
          name:this.lineInfo[i].name,
          type:'bar',
          //stack: '总量',
          areaStyle: {},
          data:series[this.lineInfo[i].type]
        })
        legend.push(this.lineInfo[i].name)
      }
```

```
      let lineOption = {
        title: {text: '明细数据'},
        tooltip: {trigger: 'axis'},
        legend: {data:legend},
        //grid: {left: '2%',right: '2%', bottom: '2%',containLabel: true},
        xAxis: {type: 'category',boundaryGap: true,data: name},
        yAxis: {type: 'value'},
        series: data
      }
      this.pie = pie
      this.$refs['lineChart'].setOptions(lineOption)
    }
  }
}
</script>

<style rel="stylesheet/scss" lang="scss" scoped>
.detail-container {
  background-color: #ffffff;
  text-align: left;
  border: 1px solid #e7e7e9;
  header {
    color: #323745;
    font-size: 14px;
    height: 55px;
    line-height: 55px;
    padding: 0 15px;
    background-color: #fbfbfb;
    border-bottom: 1px solid #e8e8e8;
  }
  .filter {
    font-size: 14px;
    padding: 20px 0 20px 20px;
    span {
      border: 1px solid #3296fa;
      color: #3296fa;
      padding: 5px 10px;
      cursor: pointer;
      &:nth-child(1){
        border-right: none;
      }
      &:nth-child(2){
        border-right: none;
      }
      &.active {
        background-color: #3296fa;
        color: #ffffff;
      }
    }
    .el-date-editor {
      margin-left: 20px;
    }
  }
  .chart {
    padding: 0 20px;
    overflow: hidden;
    margin-bottom: 30px;
  }
```

```
}
</style>
```

5.4.4 实现粉丝概况

（1）统计组件定义

在src/views/fans/components/index/中定义组件Statist.vue

```
<template>
  <div class="content">
    <el-row :gutter="40">
      <el-col :span="8">
        <div class="grid-content">
          <i></i>
          <div>
            <div>{{article}} 个</div>
            <span>图文发布量</span>
          </div>
        </div>
      </el-col>
      <el-col :span="8">
        <div class="grid-content">
          <i></i>
          <div>
            <div>{{likes}} 个</div>
            <span>点赞数量</span>
          </div>
        </div>
      </el-col>
      <el-col :span="8">
        <div class="grid-content">
          <i></i>
          <div>
            <div>{{collection}}</div>
            <span>收藏量</span>
          </div>
        </div>
      </el-col>
    </el-row>
  </div>
</template>

<script>
export default {
  props:["collection","likes","article"],
  components: {
  },
  computed: {

  },
  methods: {

  }
}
</script>
```

```
<style rel="stylesheet/scss" lang="scss" scoped>
.content {
  padding: 0 20px 20px;
}
.el-row {
  .el-col {
    .grid-content {
      background-color: #f8f8f8;
      padding: 30px 0;
      display: flex;
      justify-content: center;
      i {
        width: 50px;
        height: 50px;
        background-color: red;
        border-radius: 50%;
      }
      & > div {
        margin: 0px 0 0 10px;
        span {
          color: #444444;
          font-size: 14px;
        }
        div {
          font-size: 24px;
        }
      }
    }
  }
}

</style>
```

（2）线形图组件定义

在 src/views/fans/components/index/中定义LineChart.vue

```
<template>
  <div class="chart-content">
    <div ref="chart" :style="{height:height,width:width}"/>
  </div>
</template>

<script>
import echarts from 'echarts'
require('echarts/theme/macarons') // echarts theme

export default {
  props: {
    className: {
      type: String,
      default: 'chart'
    },
    width: {
      type: String,
      default: '100%'
    },
```

```
    height: {
      type: String,
      default: '350px'
    },
    autoResize: {
      type: Boolean,
      default: true
    },
    chartData: {
      type: Object,
      required: false
    }
  },
  data() {
    return {
      chart: null,
      sidebarElm: null
    }
  },
  watch: {
    chartData: {
      deep: true,
      handler(val) {
        this.setOptions(val)
      }
    }
  },
  mounted() {
    this.initChart()
    // 监听侧边栏的变化
    this.sidebarElm = document.getElementsByClassName('sidebar-container')[0]
    this.sidebarElm && this.sidebarElm.addEventListener('transitionend',
this.sidebarResizeHandler)
  },
  beforeDestroy() {
    if (!this.chart) {
      return
    }
    if (this.autoResize) {
      window.removeEventListener('resize', this.__resizeHandler)
    }

    this.sidebarElm && this.sidebarElm.removeEventListener('transitionend',
this.sidebarResizeHandler)

    this.chart.dispose()
    this.chart = null
  },
  methods: {
    sidebarResizeHandler(e) {
      if (e.propertyName === 'width') {
        this.__resizeHandler()
      }
    },
    setOptions(lineOption) {
      lineOption = lineOption?lineOption:{
        xAxis: {
          type: 'category',
```

```
          boundaryGap: false,
          data: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
        },
        yAxis: {
          type: 'value'
        },
        series: [{
          data: [820, 932, 901, 934, 1290, 1330, 1320],
          type: 'line',
          areaStyle: {}
        }]
      }
      this.chart.setOption(lineOption)
    },
    initChart() {
      this.chart = echarts.init(this.$refs.chart, 'macarons')
      this.setOptions()
    }
  }
}
</script>

<style rel="stylesheet/scss" lang="scss" scoped>
  .chart-content {
    padding: 0 30px;
  }
</style>
```

（3）doughnut图表组件定义

在src/views/fans/components/index/中定义DoughnutChart.vue

```
<template>
  <section class="chart">
    <div class="box">
      <div ref="chart" :style="{height:height,width:width}" class="doughnut"/>
    </div>
  </section>
</template>

<script>
import echarts from "echarts";
require("echarts/theme/macarons"); // echarts theme
export default {
  props: {
    width: {
      type: String,
      default: "100%"
    },
    height: {
      type: String,
      default: "380px"
    },
    data : {
      type:Object
    }
  },
```

```
  data() {
    return {
      chart: null
    };
  },
  watch:{
    data:function(val){
      this.initChart()
    }
  },
  mounted() {
    this.initChart();
  },
  beforeDestroy() {
    if (!this.chart) {
      return;
    }
    this.chart.dispose();
    this.chart = null;
  },
  methods: {
    initChart() {
      this.chart = echarts.init(this.$refs.chart, "macarons");
      let temp = this.data.legend
      if(temp.length>13){
        temp = temp.splice(0,14)
      }
      this.chart.setOption({
        title: {text: this.data.title},
        tooltip: {trigger: "item",formatter: "{a} <br/>{b}: {c} ({d}%)"},
        legend: {y: 'bottom',data:temp},
        series: [
          {
            name: this.data.title,
            type: "pie",
            radius: ["50%","70%"],
            roseType: 'radius',
            avoidLabelOverlap: true,
            itemStyle:{
              color: function(){return "#"+("00000"+
((Math.random()*16777215+0.5)>>0).toString(16)).slice(-6); }
            },
            label: {
              normal: {show: false,position: "center"},
              emphasis: {show: true,textStyle: {fontSize: "30",fontWeight:
"bold"}}
            },
            labelLine: {normal: {show: false}},
            data: this.data.data
          }
        ]
      });
    }
  }
};
</script>
<style rel="stylesheet/scss" lang="scss" scoped>
.chart {
```

```
  margin-top: 10px;
  width: 50%;
  float: left;
  font-size: 14px;
  .box {
    background-color: #fbfbfb;
    overflow: hidden;
    .doughnut {
      float: left;
    }
  }
  .legend {
    margin: 0 10px 10px 10px;
    margin-left: 60%;
  }
}
</style>
```

（4）数据统计整体实现

在src/views/fans/中定义index.vue

```
<template>
  <div class="fans-container">
    <div class="detail-container">
      <header>详情分析</header>
      <div class="filter">
        <el-radio-group  @change="loadDataByButton" v-model="parms.type">
          <el-radio-button  label="0">今日</el-radio-button>
          <el-radio-button  label="1">本周</el-radio-button>
          <el-radio-button  label="7">近7天</el-radio-button>
          <el-radio-button  label="30">近30天</el-radio-button>
        </el-radio-group>
        <el-date-picker v-model="parms.time" type="datetimerange"
                        range-separator="-"
                        start-placeholder="开始日期"
                        end-placeholder="结束日期"  @change="loadDataByTimeRange"
:picker-options="pickerOptions" format="yyyy-MM-dd HH:mm:ss" placeholder="选择日
期"/>
      </div>
      <Statist :article="all.article" :likes="all.likes"
:collection="all.collection"/>
      <line-chart ref="lineChart"/>
      <div class="chart">
        <template v-for="item in pie">
          <doughnut-chart :data="item" v-if="item.title !='发文量-转化率'"/>
        </template>
      </div>
    </div>
  </div>
</template>

<script>
import Statist from './components/index/Statist.vue'
import LineChart from './components/index/LineChart.vue'
import DoughnutChart from './components/index/DoughnutChart.vue'
import {getFansStatistics} from '@/api/fans'
```

```
import DateUtil from '@/utils/date'

export default {
  name: 'ContentManage',
  data() {
    return {
      parms:{
        type:'0',
        stime:'',
        etime:''
      },
      all:{},
      list:'',
      graph:'',
      pie:{},
      lineInfo : [
        {name:'发文量',type:'article'},
        {name:'阅读量',type:'read_count'},
        {name:'点赞量',type:'likes'},
        {name:'评论量',type:'comment'},
        {name:'收藏量',type:'collection'},
        {name:'转发量',type:'follow'},
        {name:'不喜欢',type:'unlikes'}
      ],
      pickerOptions: {
        disabledDate(time) {
          return time.getTime() > Date.now()
        }
      }
    }
  },
  components: {
    Statist,
    LineChart,
    DoughnutChart
  },
  created(){
    this.getFansStatistics()
  },
  methods : {
    loadDataByTimeRange:function(e){
      this.parms.type=-1
      this.parms.stime=e[0].getTime()
      this.parms.etime=e[1].getTime()
      this.getFansStatistics();
    },
    loadDataByButton:function(e){
      if(e=='1'){// 本周
        this.parms.stime=DateUtil.getWeekSTime()
        this.parms.etime=DateUtil.getWeekETime()
      }else{
        this.parms.stime=DateUtil.getNearTime(0)
        this.parms.etime=DateUtil.getNearTime(e)
      }
      this.getFansStatistics();
    },
    async getFansStatistics (){
      console.log(this.parms)
```

```
      let result = await getFansStatistics(this.parms)
      this.list = result.data
      let all =
{article:0,likes:0,collection:0,forward:0,comment:0,read_count:0}
      let chats = {}
      for (let i = 0; i < result.data.length; i++) {
        let tmp = result.data[i];
        let time = DateUtil.format13(tmp.created_time)
        let data = chats[time]?chats[time]:{}
        for (let j = 0; j <this.lineInfo.length ; j++) {
          let k=this.lineInfo[j].type
          all[k]+=tmp[k]
          let val = data[k]?data[k]:0
          val +=tmp[k]
          data[k]=val
        }
        chats[time]=data
      }
      this.all = all
      this.graph = chats
      this.parseToLine(chats,all)
    },
    parseToLine : function(chats,all){
      // 排序
      var name = [];
      for (let k in chats) {
        name.push(k)
      }
      name.sort()
      let series = {}//折线图数据
      let pie = {}//饼图数据
      for (let i = 0; i <name.length ; i++) {
        for (let j = 0; j <this.lineInfo.length ; j++) {
          let k=this.lineInfo[j].type
          series[k] = series[k]?series[k]:[]
          series[k].push(chats[name[i]][k])
          pie[k] = pie[k]?pie[k]:{}
          pie[k]['title'] = this.lineInfo[j].name+'-转化率'
          pie[k]['data'] = pie[k]['data']?pie[k]['data']:[]
          pie[k]['legend'] = pie[k]['legend']?pie[k]['legend']:[]
          pie[k]['legend'].push(name[i])
          //pie[k]['data'].push({value:(chats[name[i]][k]/chats[name[i]]
['article']).toFixed(2),name:name[i]})
          pie[k]['data'].push({value:chats[name[i]][k],name:name[i]})
        }
      }
      let data = []
      let legend=[]
      for (let i = 0; i <this.lineInfo.length ; i++) {
        data.push({
          name:this.lineInfo[i].name,
          type:'line',
          //stack: '总量',
          areaStyle: {},
          data:series[this.lineInfo[i].type]
        })
        legend.push(this.lineInfo[i].name)
      }
```

```
      let lineOption = {
        title: {text: '明细数据'},
        tooltip: {trigger: 'axis'},
        legend: {data:legend},
        //grid: {left: '2%',right: '2%', bottom: '2%',containLabel: true},
        xAxis: {type: 'category',boundaryGap: false,data: name},
        yAxis: {type: 'value'},
        series: data
      }
      this.pie = pie
      this.$refs['lineChart'].setOptions(lineOption)
    }
  }
}
</script>

<style rel="stylesheet/scss" lang="scss" scoped>
.fans-container {
  background-color: #ffffff;
  text-align: left;
  border: 1px solid #e7e7e9;
  header {
    color: #323745;
    font-size: 14px;
    height: 55px;
    line-height: 55px;
    padding: 0 15px;
    background-color: #fbfbfb;
    border-bottom: 1px solid #e8e8e8;
  }
  .tabBar {
    font-size: 14px;
    padding: 0 15px;
    height: 55px;
    line-height: 55px;
    border-bottom: 1px dashed #cccccc;
    a {
      margin-right: 35px;
      color: #323745;
      &.active {
        color: #3296fa;
      }
    }
  }
  .filter {
    font-size: 14px;
    padding: 20px 0 20px 20px;
    span {
      border: 1px solid #3296fa;
      color: #3296fa;
      padding: 5px 10px;
      cursor: pointer;
      &:nth-child(1){
        border-right: none;
      }
      &:nth-child(2){
        border-right: none;
      }
```

```
      &.active {
        background-color: #3296fa;
        color: #ffffff;
      }
    }
    .el-date-editor {
      margin-left: 20px;
    }
  }
  .chart {
    padding: 0 20px;
    overflow: hidden;
    margin-bottom: 30px;
  }
}
</style>
```