项目优化

目标

- 能够完成图片微服务开发及应用
- 改造app文章查询列表，主推热点文章数据
- 能够掌握联想词优化改造的思路及trie树算法的应用
- 能够改造登录接口，进行秘密验证

1 图片缓存开发

1.1 功能需求

随着热文章的访问越来越频繁，文章内的图片频繁的访问图片服务器，对图片服务器造成了很大的压力，因此决定把热文章相关的图片缓存到redis：

- 监听热文章主消息缓存图片功能

本案例开发功能包括：

- 监听热文章消息缓存图片

1.2 article微服务发送消息

1.2.1 定义消息名称 common模块

maven_test.properties

```
kafka.topic.hot-article=heima.topic.hot.article.sigle.test
```

kafka.properties

```
kafka.topic.hot-article=${kafka.topic.hot-article}
```

读取消息名称，修改com.heima.common.kafka.KafkaTopicConfig

```
String hotArticle;
```

1.2.2 定义封装实体类

创建类：com.heima.common.kafka.messages.app.ApHotArticleMessage

```
public class ApHotArticleMessage extends KafkaMessage<ApHotArticles> {

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

1.2.3 消息发送

修改com.heima.common.kafka.KafkaSender,添加方法

```
/**
     * 发送处理热文章信息
     *
     * @param message
     */
public void sendHotArticleMessage(ApHotArticles message) {
    ApHotArticleMessage temp = new ApHotArticleMessage();
    temp.setData(message);
    this.sendMesssage(kafkaTopicConfig.getHotArticle(),
UUID.randomUUID().toString(), temp);
}
```

1.2.4 修改计算热点文章的代码

修改类：com.heima.article.service.impl.ApHotArticleServiceImpl

```
@Autowired
private KafkaSender kafkaSender;

@Override
public void computeHotArticle() {
    //...代码省略...
    for (ApHotArticles hot : hotArticlesList) {
        //保存热点文章
        apHotArticlesMapper.insert(hot);
        //给每一个用户保存热点数据
        saveHotArticleForEntryList(hot,entryList);
        //缓存文章中的图片
         kafkaSender.sendHotArticleMessage(hot);
    }
}
```

1.3 images微服务接收热点文章，处理图片

导入资料中模块heima-leadnews-images 专门处理图片相关的微服务 集成redis

工具类：

创建：com.heima.utils.common.Base64Utils

```
public class Base64Utils {

    /**
```

```
     * 解码
     * @param base64
     * @return
     */
    public static byte[] decode(String base64){
        BASE64Decoder decoder = new BASE64Decoder();
        try {
            // Base64解码
            byte[] b = decoder.decodeBuffer(base64);
            for (int i = 0; i < b.length; ++i) {
                if (b[i] < 0) {// 调整异常数据
                    b[i] += 256;
                }
            }
            return b;
        } catch (Exception e) {
            return null;
        }
    }


    /**
     * 编码
     * @param data
     * @return
     * @throws Exception
     */
    public static String encode(byte[] data) {
        BASE64Encoder encoder = new BASE64Encoder();
        return encoder.encode(data);
    }
}
```

思路分析：

- 监听article微服务的消息，获取热点文章数据
- 获取文章的content，解析出里面的图片，进行redis缓存
- 获取文章封面图片，进行redis缓存

注意:只缓存fastdfs中的图片信息

（2）创建com.heima.images.service.CacheImageService 缓存图片

```
public interface CacheImageService {

    /**
     * 缓存图片到redis
     * @param imgUrl
     */
    byte[] cache2Redis(String imgUrl, boolean isCache);

    /**
     * 延长图片缓存
     * @param imageKey
     */
    void resetCache2Redis(String imageKey);
}
```

(3) 创建接口处理热点文章图片数据com.heima.images.service.HotArticleImageService

```
public interface HotArticleImageService {

    /**
     * 处理热文章消息
     * @param message
     */
    public void handleHotImage(ApHotArticleMessage message);
}
```

(4) 创建实现类com.heima.images.service.impl.CacheImageServiceImpl

```
@Service
@Log4j2
public class CacheImageServiceImpl implements CacheImageService {

    @Autowired
    private FastDfsClient fastDfsClient;
    @Autowired
    private StringRedisTemplate redisTemplate;

    final long EXPIRE = 60 * 60 * 24l; //24小时

    @Override
    public byte[] cache2Redis(String imgUrl, boolean isCache) {
        byte[] ret = null;
        log.info("缓存图片到redis#imgUrl:{},isCache:{}", imgUrl, isCache);
        //http://47.94.7.85/group1/M00/00/00/rBENvl02ZtKAEgFqAACNdiGk7IM981.jpg
        Map<String,String> map = formatPath(imgUrl);
        String group = map.get("group");
        String file = map.get("file");
        String baseString = "";
        try {
            byte[] fileByte = fastDfsClient.downGroupFile(group, file);
            ret = fileByte;
            baseString = Base64Utils.encode(fileByte);
        } catch (Exception e) {
            e.printStackTrace();
        }
        if(isCache){
            redisTemplate.opsForValue().set(imgUrl, baseString, EXPIRE,
TimeUnit.SECONDS);
        }
        return ret;
    }

    @Override
    public void resetCache2Redis(String imageKey) {
        redisTemplate.expire(imageKey, EXPIRE, TimeUnit.SECONDS);
    }

    /**
     * 解析图片URL
     * @param imgUrl
     * @return
     */
```

```
    private Map<String,String> formatPath(String imgUrl){
        Map<String,String> map = Maps.newHashMap();
        String groupString =
imgUrl.substring(imgUrl.indexOf("group"),imgUrl.length());
        int index = groupString.indexOf("/");
        map.put("group", groupString.substring(0,index));
        map.put("file", groupString.substring(index+1,groupString.length()));
        return map;
    }
}
```

(5)创建实现类com.heima.images.service.impl.HotArticleImageServiceImpl

```
@Service
@Log4j2
public class HotArticleImageServiceImpl implements HotArticleImageService {


    @Autowired
    private ApArticleContentMapper apArticleContentMapper;
    @Autowired
    private CacheImageService cacheImageService;
    @Autowired
    private ApArticleMapper apArticleMapper;

    @Override
    public void handleHotImage(ApHotArticleMessage message) {
        ApHotArticles hotArticles =  message.getData();
        log.info("处理热文章图片开始#articleId:{},message:{}",
hotArticles.getArticleId(), JSON.toJSONString(message));
        ApArticleContent content =
apArticleContentMapper.selectByArticleId(hotArticles.getArticleId());
        //文章内容缓存
        String source = ZipUtils.gunzip(content.getContent());
        JSONArray array = JSONArray.parseArray(source);
        for (int i = 0; i< array.size(); i++) {
            JSONObject obj = array.getJSONObject(i);
            if(!"image".equals(obj.getString("type"))) continue;
            String imagePath = obj.getString("value");
            if(!imagePath.startsWith(InitConfig.PREFIX)){
                log.info("非站内图片不缓存#articleId:{}",
hotArticles.getArticleId());
                continue;
            }
            //缓存图片
            cacheImageService.cache2Redis(imagePath, true);
        }
        //主图缓存
        ApArticle article =
apArticleMapper.selectById(Long.valueOf(hotArticles.getArticleId()));
        if(StringUtils.isNotEmpty(article.getImages())){
            String[] articleLmages = article.getImages().split(",");
            for (String img : articleLmages){
                if(!img.startsWith(InitConfig.PREFIX)){
                    log.info("非站内图片不缓存#articleId:{}",
hotArticles.getArticleId());
                    continue;
```

```
                }
                //缓存图片
                cacheImageService.cache2Redis(img, true);
            }
        }
        log.info("处理热文章图片结束#message:{}", JSON.toJSONString(message));
    }
}
```

(6)创建消息监听类，监听消息

com.heima.images.kafka.listener.HotArticleListener

```
@Component
@Log4j2
public class HotArticleListener implements
com.heima.common.kafka.KafkaListener<String,String> {

    @Autowired
    KafkaTopicConfig kafkaTopicConfig;
    @Autowired
    ObjectMapper mapper;
    @Autowired
    HotArticleImageService hotArticleImageService;

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

    @Override
    public void onMessage(ConsumerRecord<String, String> data, Consumer<?, ?>
consumer) {
        log.info("receive hot article message:{}",data);
        String value = (String)data.value();
        try {
            ApHotArticleMessage message = mapper.readValue(value,
ApHotArticleMessage.class);
            hotArticleImageService.handleHotImage(message);
        }catch (Exception e){
            log.error("kafka send message[class:{}] to handleHotImage failed:
{}","ApHotArticleMessage.class",e);
        }
    }
}
```

1.4 查询图片接口

1.4.1 基本定义

参考标准	请参考通用接口规范
接口名称	/api/v1/images/get
请求DTO	String
响应DTO	BufferedImage

1.4.2 code定义

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

1.4.3 代码实现

创建类：com.heima.images.apis.ImagesControllerApi

此类在apis模块中创建，定义了相关接口，实现如下：

```
/**
 * 访问缓存图片
 * @param imagePath
 * @return
 * @throws Exception
 */
public BufferedImage getImage(String imagePath) throws Exception;
```

创建类：com.heima.images.controller.v1.ImagesController

```
@Controller
@RequestMapping(value = "api/v1/images")
@Log4j2
public class ImagesController {

    @Autowired
    private StringRedisTemplate redisTemplate;

    @Autowired
    private CacheImageService cacheImageService;


    @RequestMapping(value = "get",
            produces = MediaType.IMAGE_JPEG_VALUE,
            method = RequestMethod.GET)
    @ResponseBody
    public BufferedImage getImage(String u) throws Exception {
        String path = u;
        if(!u.startsWith("http")){
            path = InitConfig.PREFIX+u;
        }
        log.info("图片访问请求开始#path:{}", path);
        String baseCode = redisTemplate.opsForValue().get(path);
        //不存在从fds中读取
        if(StringUtils.isEmpty(baseCode)){
```

```
            byte[] cache = cacheImageService.cache2Redis(path, false);
            BufferedImage bufferedImage = ImageIO.read(new
ByteArrayInputStream(cache));
            return bufferedImage;
        }
        byte[] source = Base64Utils.decode(baseCode);
        BufferedImage bufferedImage = ImageIO.read(new
ByteArrayInputStream(source));
        log.info("图片访问请求结束#path:{}", path);
        return bufferedImage;
    }
}
```

2 文章首页接口改造

2.1 需求分析

添加v2接口，首页列表首先加载热点文章，按照频道从缓存获取首页数据

app端图片优先从缓存中读取

2.2 后端改造

2.1.1 mapper

修改ApArticleMapper，添加方法

```
/**
     * 依据文章IDS来获取文章详细内容
     * @param list 文章ID
     * @return
     */
List<ApArticle> loadArticleListByIdListV2(List<Integer> list);
```

ApArticleMapper.xml

```
<!-- 以及文章IDS列表获取文章数据V2 -->
<select id="loadArticleListByIdListV2" resultMap="resultMap">
    select * from ap_article where id in(
    <trim prefix="" suffixOverrides=",">
        <foreach item="item" collection="list" separator=",">
            #{item}
        </foreach>
    </trim>
    )
</select>
```

2.1.2 service

修改类com.heima.article.service.AppArticleService，添加v2版本的方法

```
/**
     * 加载文章列表数据
     * @param type 1 加载更多  2 加载更新
     * @param dto 封装数据
     * @return 数据列表
     */
ResponseResult loadV2(Short type, ArticleHomeDto dto, boolean firstPage);
```

实现类：com.heima.article.service.impl.AppArticleServiceImpl

```
@Autowired
private ApHotArticlesMapper apHotArticlesMapper;

@Autowired
private StringRedisTemplate redisTemplate;

@Autowired
private ApBehaviorEntryMapper apBehaviorEntryMapper;

@Override
public ResponseResult loadV2(Short type, ArticleHomeDto dto, boolean firstPage)
{
    if(null == dto){
        dto = new ArticleHomeDto();
    }
    ApUser user = AppThreadLocalUtils.getUser();
    Integer size = dto.getSize();
    String tag = dto.getTag();
    // 分页参数校验
    if (size == null || size <= 0) {
        size = 20;
    }
    size = Math.min(size,MAX_PAGE_SIZE);
    dto.setSize(size);
    //  类型参数校验
    if (!type.equals(ArticleConstans.LOADTYPE_LOAD_MORE) &&
!type.equals(ArticleConstans.LOADTYPE_LOAD_NEW))
        type = ArticleConstans.LOADTYPE_LOAD_MORE;
    // 文章频道参数验证
    if (StringUtils.isEmpty(tag)) {
        dto.setTag(ArticleConstans.DEFAULT_TAG);
    }
    // 最大时间处理
    if(dto.getMaxBehotTime()==null){
        dto.setMaxBehotTime(new Date());
    }
    // 最小时间处理
    if(dto.getMinBehotTime()==null){
        dto.setMinBehotTime(new Date());
    }
    //从缓存中读取 否则数据库查询
    if(firstPage){
        // 数据加载
        List<ApArticle> cacheList = getCacheArticleV2(dto);
        if(cacheList.size()>0){
            log.info("使用缓存加载数据#tag:{}", dto.getTag());
```

```
            return ResponseResult.okResult(getCacheArticleV2(dto));
        }
    }
    // 数据加载
    if(user!=null){
        return ResponseResult.okResult(getUserArticleV2(user,dto,type));
    }else{
        return ResponseResult.okResult(getDefaultArticleV2(dto,type));
    }
}
/**
     * 查询缓存首页文章数据
     * @param dto
     * @return
     */
private List<ApArticle> getCacheArticleV2(ArticleHomeDto dto) {
    log.info("查询缓存热文章数据#tag:{}", dto.getTag());
    String key = ArticleConstans.HOT_ARTICLE_FIRST_PAGE + dto.getTag();
    String ret = redisTemplate.opsForValue().get(key);
    if(StringUtils.isEmpty(ret)){
        return Lists.newArrayList();
    }
    List<ApArticle> list = JSONArray.parseArray(ret, ApArticle.class);
    log.info("查询缓存热文章数据#tag:{}, size:{}", dto.getTag(), list.size());
    return list;
}
/**
     * 先从用户的推荐表中查找文章，如果没有再从大文章列表中获取
     * @param user
     * @param dto
     * @param type
     * @return
     */
private List<ApArticle> getUserArticleV2(ApUser user, ArticleHomeDto dto, Short
type){
    // 用户和设备不能同时为空
    if(user == null){
        return Lists.newArrayList();
    }
    Long userId = user.getId();
    ApBehaviorEntry apBehaviorEntry =
apBehaviorEntryMapper.selectByUserIdOrEquipemntId(userId, null);
    // 行为实体找以及注册了，逻辑上这里是必定有值得，除非参数错误
    if(apBehaviorEntry==null){
        return Lists.newArrayList();
    }
    Integer entryId =  apBehaviorEntry.getId();
    //如果没查到 查询全局热文章
    if(entryId==null)entryId=0;
    long time = System.currentTimeMillis();
    List<ApHotArticles> list =
apHotArticlesMapper.loadArticleIdListByEntryId(entryId, dto, type);
    System.out.println("==================1=:"+(System.currentTimeMillis()-
time));
    //默认从热文章里查询
    if(!list.isEmpty()){
        List<Integer> articleList = list.stream().map(p ->
p.getArticleId()).collect(Collectors.toList());
```

```
        List<ApArticle> temp =
apArticleMapper.loadArticleListByIdListV2(articleList);
        System.out.println("==================2=:"+(System.currentTimeMillis()-
time));
        return temp;
    }else{
        return getDefaultArticleV2(dto,type);
    }
}
/**
     * 从默认的热数据列表中获取文章
     * @param dto
     * @param type
     * @return
     */
private List<ApArticle> getDefaultArticleV2(ArticleHomeDto dto,Short type){
    List<ApHotArticles> hotList = apHotArticlesMapper.loadHotListByLocation(dto,
type);
    List<ApArticle> articleList = Lists.newArrayList();
    for (ApHotArticles hot: hotList) {
        ApArticle article =
apArticleMapper.selectById(Long.valueOf(hot.getArticleId()));
        articleList.add(article);
    }
    return articleList;
}
```

2.1.3 controller

新建类：com.heima.article.controller.v2.ArticleHomeV2Controller

```
@RestController
@RequestMapping("/api/v2/article")
public class ArticleHomeV2Controller implements ArticleHomeControllerApi {

    @Autowired
    private AppArticleService appArticleService;

    @Override
    @PostMapping("/load")
    public ResponseResult load(@RequestBody ArticleHomeDto dto) {
        return appArticleService.loadV2( ArticleConstans.LOADTYPE_LOAD_MORE,
dto, true);
    }

    @Override
    @PostMapping("/loadmore")
    public ResponseResult loadMore(@RequestBody ArticleHomeDto dto) {
        return appArticleService.loadV2( ArticleConstans.LOADTYPE_LOAD_MORE,
dto, false);
    }

    @Override
    @PostMapping("/loadnew")
    public ResponseResult loadNew(@RequestBody ArticleHomeDto dto) {
        return appArticleService.loadV2( ArticleConstans.LOADTYPE_LOAD_NEW, dto,
false);
```

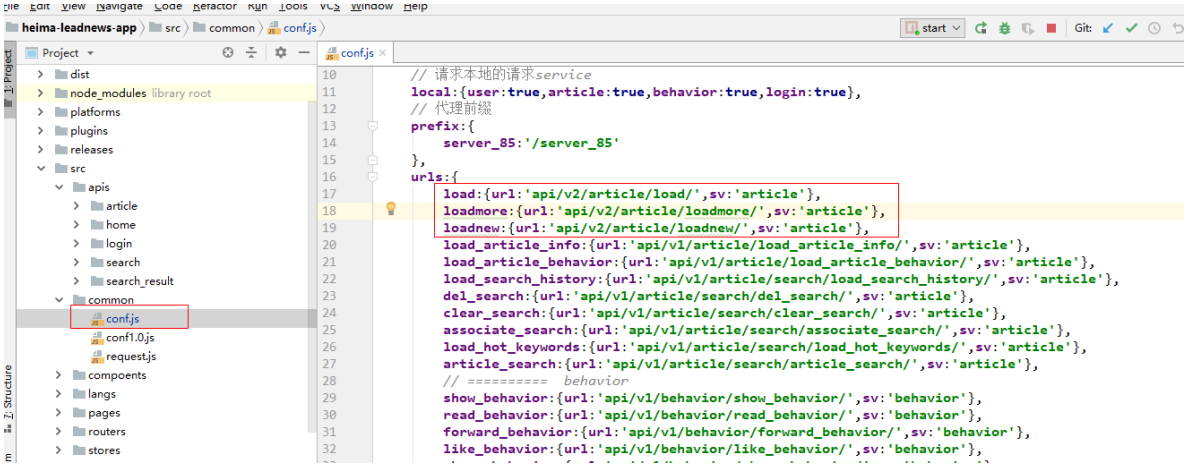
```
    }

}
```

2.2 前端改造

2.2.1 文章列表加载

修改src/common/conf.js

效果如下，修改接口地址为v2



2.2.2 图片访问

修改src\compoents\cells\article_1.vue和article_3.vue

```
<template>
    <div class="list-item">
        <div class="list-lr">
            <div class="item-l">
                <text class="title">{{formatTitle(data.title)}}</text>
                <div class="tags">
                    <text class="tags-text tags-icon">{{data.icon}}</text>
                    <text class="tags-text">{{data.source}}</text>
                    <text class="tags-text">评论 {{data.commit}}</text>
                </div>
            </div>
            <div class="item-r">
                <image class="image" :src="formatImg(data.image[0])"/>
            </div>
        </div>
    </div>
</template>

<script>
    export default {
        name: "article_1",
        props:{
            data:{
                type:Object
            }
        },
        methods : {
```

```
            formatDate:function(time){
                return this.$date.format13(time);
            },formatTitle:function(title){
                if(title.length>32){
                    return title.substring(0,31);
                }
                return title;
            },formatImg:function (img) {
                if(img.indexOf('group')){
                    return "http://localhost:9010/api/v1/images/get?u="+img;
                }else{
                    return img;
                }
            }
        }
    }
</script>

<style lang="less" scoped>
    @import '../../styles/article';
    .list-item{
    }
    .list-lr{
        flex-direction: row;
        justify-content: space-around;
    }
    .item-l{
        width: 550px;
    }
    .item-r{
        width: 180px;
        margin: @list-tb-margin @list-lr-margin;
    }
</style>
```

3 联想词优化

3.1 功能需求

随着联想词的搜索越来越频繁，每次从数据库查询非常占用数据库资源，同时查询效率比较低，因此对联想词查询功能做改造：

- 数据能够缓存到redis
- 构造Trie树数据结构，高效查询数据

缓存联想词数据，使用Trie数据结构优化查询效率。

3.2 名词解释

Trie树：又称单词查找树，Trie树，是一种树形结构，是一种哈希树的变种。典型应用是用于统计，排序和保存大量的字符串（但不仅限于字符串），所以经常被搜索引擎系统用于文本词频统计。它的优点是：利用字符串的公共前缀来减少查询时间，最大限度地减少无谓的字符串比较，查询效率比哈希树高。

3.3 接口定义

查询联想词接口V2

3.3.1 基本定义

参考标准	请参考通用接口规范
接口名称	/api/v2/article/search/associate_search
请求DTO	com.heima.model.article.dtos.UserSearchDto
响应DTO	ResponseResult

3.3.2 CODE定义

PARAM_INVALID	PARAM_INVALID(501,"无效参数"),
AP_USER_DATA_NOT_EXIST	AP_USER_DATA_NOT_EXIST(1000,”ApUser数据不存在”),
LOGIN_PASSWORD_ERROR	LOGIN_PASSWORD_ERROR(2,”密码错误”),

3.3.3 类定义说明

- ApUser是对应数据表的POJO对象，放置model模块
- ApUserMapper是MybatisMapper文件，放置在model模块
- LoginControllerApi是服务接口定义，放置在apis模块
- ApUserLoginService、ApUserLoginServiceImpl、LoginController是对功能的实现，放置在login模块

3.3.4 Mapper实现

（1）ApAssociateWordsMapper

定义查询所有联想词方法：

```
/**
 * 查询联想词
 * @return
 */
List<ApAssociateWords> selectAllAssociateWords();
```

ApAssociateWordsMapper.xml

```
<select id="selectAllAssociateWords" resultMap="BaseResultMap" >
    select
    <include refid="Base_Column_List" />
    from ap_associate_words
</select>
```

3.3.5 时序说明

- 判断参数是否服务，否则返回PARAM_INVALID（无效参数）
- 查询所有的联想词，如果缓存有从缓存中获取
- 创建Trie字典树
- 从字典树中搜索
- 返回数据

3.3.6 工具类说明

com.heima.article.utils.TrieNode

```
public class TrieNode {
    public char var;
    public boolean isWord;
    public Map<Character,TrieNode> children = new HashMap<>();
    public boolean containLongTail = false;
    public TrieNode(){}
    public TrieNode(char c){
        TrieNode node = new TrieNode();
        node.var = c;
    }
}
```

创建工具类：com.heima.article.utils.Trie

创建字典树工具类：

```
public class Trie {
    private TrieNode root;
    public Trie(){
        root = new TrieNode();
        root.var = ' ';
    }
    /**
     * 插入trie树
     * @param word
     */
    public void insert(String word){
        TrieNode ws = root;
        for(int i = 0; i < word.length(); i++){
            char c = word.charAt(i);
            if(!ws.children.keySet().contains(c)){
                ws.children.put(c,new TrieNode(c));
            }
            ws = ws.children.get(c);
        }
        ws.isWord = true;
    }

    /**
     * 查询trie树
     * @param prefix
     * @return
     */
    public List<String> startWith(String prefix){
```

```
        List<String> match = new ArrayList<>();
        TrieNode ws = root;
        for(int i = 0; i < prefix.length(); i++){
            char c = prefix.charAt(i);
            if(!ws.children.keySet().contains(c)) return match;
            ws = ws.children.get(c);
            if(!ws.containLongTail){
                for (char cc : ws.children.keySet()){
                    match.add(prefix+cc);
                }
            }else{
                //包含长尾词 从map中取
            }
        }
        return match;
    }
}
```

3.3.7 代码实现

（1）ApArticleSearchService

```
/**
 * 联想词V2
 * @param userSearchDto
 * @return
 */
ResponseResult searchAssociateV2(UserSearchDto userSearchDto);
```

（2）ApArticleSearchServiceImpl

实现关联查询接口

```
@Autowired
private StringRedisTemplate redisTemplate;

@Override
public ResponseResult searchAssociateV2(UserSearchDto dto) {
    if(dto.getPageSize()>50){
        return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_INVALID);
    }
    String assoStr = redisTemplate.opsForValue().get("associate_list");
    List<ApAssociateWords> aw = null;
    if(StringUtils.isNotEmpty(assoStr)){
        aw = JSON.parseArray(assoStr, ApAssociateWords.class);
    }else{
        aw = apAssociateWordsMapper.selectAllAssociateWords();
        redisTemplate.opsForValue().set("associate_list",
JSON.toJSONString(aw));
    }
    //needed cache trie
    Trie t = new Trie();
    for (ApAssociateWords a : aw){
        t.insert(a.getAssociateWords());
    }
    List<String> ret = t.startWith(dto.getSearchWords());
```

```
    List<ApAssociateWords> wrapperList = Lists.newArrayList();
    for(String s : ret){
        ApAssociateWords apAssociateWords = new ApAssociateWords();
        apAssociateWords.setAssociateWords(s);
        wrapperList.add(apAssociateWords);
    }
    return ResponseResult.okResult(wrapperList);
}
```

v2版本的controller

```
@RestController
@RequestMapping("/api/v2/article/search")
public class ArticleSearchV2Controller implements ArticleSearchControllerApi {

    @Autowired
    private ApArticleSearchService apArticleSearchService;

    @PostMapping("load_search_history")
    @Override
    public ResponseResult findUserSearch(@RequestBody UserSearchDto
userSearchDto) {
        return apArticleSearchService.findUserSearch(userSearchDto);
    }

    @PostMapping("del_search")
    @Override
    public ResponseResult delUserSearch(@RequestBody UserSearchDto
userSearchDto) {
        return apArticleSearchService.delUserSearch(userSearchDto);
    }

    @PostMapping("clear_search")
    @Override
    public ResponseResult clearUserSearch(@RequestBody UserSearchDto
userSearchDto) {
        return apArticleSearchService.clearUserSearch(userSearchDto);
    }

    @PostMapping("associate_search")
    @Override
    public ResponseResult searchassociate(@RequestBody UserSearchDto
userSearchDto) {
        return apArticleSearchService.searchAssociateV2(userSearchDto);
    }

    @PostMapping("load_hot_keywords")
    @Override
    public ResponseResult hotkeywords(@RequestBody UserSearchDto userSearchDto)
{
        return apArticleSearchService.hotkeywords(userSearchDto.getHotDate());
    }

    @PostMapping("article_search")
    @Override
    public ResponseResult esArticleSearch(@RequestBody UserSearchDto
userSearchDto) {
```

```
        return apArticleSearchService.esArticleSearch(userSearchDto);
    }
}
```

3.3.8 前端集成测试

修改src\common\conf.js

```
    urls:{
        load:{url:'api/v2/article/load/',sv:'article'},
        loadmore:{url:'api/v2/article/loadmore/',sv:'article'},
        loadnew:{url:'api/v2/article/loadnew/',sv:'article'},
        load_article_info:{url:'api/v1/article/load_article_info/',sv:'article'},
        load_article_behavior:{url:'api/v1/article/load_article_behavior/',sv:'article'},
        load_search_history:{url:'api/v1/article/search/load_search_history/',sv:'article'},
        del_search:{url:'api/v1/article/search/del_search/',sv:'article'},
        clear_search:{url:'api/v1/article/search/clear_search/',sv:'article'},
        associate_search:{url:'api/v2/article/search/associate_search/',sv:'article'},
        load_hot_keywords:{url:'api/v1/article/search/load_hot_keywords/',sv:'article'},
        article_search:{url:'api/v1/article/search/article_search/',sv:'article'},
```

4 接口安全

4.1 功能需求

在互联网的世界里，安全是一个避不开的话题，如何让用户的信息在保护下访问应用，不暴露在互联网，被他人获取盗用，就需要了解加密相关的知识，本节从密码加密和验证这个功能带你进入这个全新的领域：

- 改造登录功能

本案例开发功能包括：

- 改造登录功能进行加密验证
- 实现不同的加密工具类

4.2 接口定义

用户登录验证接口

4.2.1 基本定义

参考标准	请参考通用接口规范
接口名称	/api/v2/login/login_auth
请求DTO	com.heima.model.user.pojos.ApUser
响应DTO	

4.2.2 CODE定义

PARAM_INVALID	PARAM_INVALID(501,"无效参数"),
AP_USER_DATA_NOT_EXIST	AP_USER_DATA_NOT_EXIST(1000,"ApUser数据不存在"),

PARAM_INVALID	PARAM_INVALID(501,"无效参数"),
LOGIN_PASSWORD_ERROR	LOGIN_PASSWORD_ERROR(2,"密码错误"),

4.2.3 工具类说明

(1)MD5Utils

创建类:com.heima.utils.common.MD5Utils

```
public class MD5Utils {

    /**
     * MD5加密
     * @param str
     * @return
     */
    public final static String encode(String str) {
        try {
            //创建具有指定算法名称的摘要
            MessageDigest md = MessageDigest.getInstance("MD5");
            //使用指定的字节数组更新摘要
            md.update(str.getBytes());
            //进行哈希计算并返回一个字节数组
            byte mdBytes[] = md.digest();
            String hash = "";
            //循环字节数组
            for (int i = 0; i < mdBytes.length; i++) {
                int temp;
                //如果有小于0的字节,则转换为正数
                if (mdBytes[i] < 0)
                    temp = 256 + mdBytes[i];
                else
                    temp = mdBytes[i];
                if (temp < 16)
                    hash += "0";
                //将字节转换为16进制后，转换为字符串
                hash += Integer.toString(temp, 16);
            }
            return hash;
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        return "";
    }

    public static String encodeWithSalt(String numStr, String salt) {
        return encode(encode(numStr) + salt);
    }

    public static void main(String[] args) {
        System.out.println(encode("123456"));
        System.out.println(encodeWithSalt("123456","123456"));;
    }
}
```

(2)DESUtils

创建工具类：com.heima.utils.common.DESUtils

```
public class DESUtils {

    public static final String key = "12345678";

    /**
     * 加密
     * @param content
     * @param keyBytes
     * @return
     */
    private static byte[] encrypt(byte[] content, byte[] keyBytes) {
        try {
            DESKeySpec keySpec = new DESKeySpec(keyBytes);
            String algorithm =  "DES";//指定使什么样的算法
            SecretKeyFactory keyFactory =
SecretKeyFactory.getInstance(algorithm);
            SecretKey key = keyFactory.generateSecret(keySpec);
            String transformation = "DES/CBC/PKCS5Padding"; //用什么样的转型方式
            Cipher cipher = Cipher.getInstance(transformation);
            cipher.init(Cipher.ENCRYPT_MODE, key, new
IvParameterSpec(keySpec.getKey()));
            byte[] result = cipher.doFinal(content);
            return result;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    /**
     * 解密
     * @param content
     * @param keyBytes
     * @return
     */
    private static byte[] decrypt(byte[] content, byte[] keyBytes) {
        try {
            DESKeySpec keySpec = new DESKeySpec(keyBytes);
            String algorithm = "DES";
            SecretKeyFactory keyFactory = SecretKeyFactory.getInstance(algorithm
);
            SecretKey key = keyFactory.generateSecret(keySpec);
            String transformation = "DES/CBC/PKCS5Padding";
            Cipher cipher = Cipher.getInstance(transformation );
            cipher.init(Cipher.DECRYPT_MODE, key, new
IvParameterSpec(keyBytes));
            byte[] result = cipher.doFinal(content);
            return result;

        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
```

```
    }


    /**
     * 二进制转16进制
     * @param bytes
     * @return
     */
    private static String byteToHexString(byte[] bytes) {
        StringBuffer sb = new StringBuffer();
        String sTemp;
        for (int i = 0; i<bytes.length; i++) {
            sTemp = Integer.toHexString(0xFF & bytes[i]);
            if (sTemp.length() < 2) {
                sb.append(0);
            }
            sb.append(sTemp.toUpperCase());
        }
        return sb.toString();
    }

    /**
     * 16进制字符串转bytes
     * @param hex
     * @return
     */
    public static byte[] hexStringToByte(String hex) {
        int len = 0;
        int num=0;
        //判断字符串的长度是否是两位
        if(hex.length()>=2){
            //判断字符喜欢是否是偶数
            len=(hex.length() / 2);
            num = (hex.length() % 2);
            if (num == 1) {
                hex = "0" + hex;
                len=len+1;
            }
        }else{
            hex = "0" + hex;
            len=1;
        }
        byte[] result = new byte[len];
        char[] achar = hex.toCharArray();
        for (int i = 0; i < len; i++) {
            int pos = i * 2;
            result[i] = (byte) (toByte(achar[pos]) << 4 | toByte(achar[pos +
1]));
        }
        return result;
    }
    private static int toByte(char c) {
        if (c >= 'a')
            return (c - 'a' + 10) & 0x0f;
        if (c >= 'A')
            return (c - 'A' + 10) & 0x0f;
        return (c - '0') & 0x0f;
    }
```

```
    private static byte[] hexToByteArr(String strIn) {
        byte[] arrB = strIn.getBytes();
        int iLen = arrB.length;
        // 两个字符表示一个字节，所以字节数组长度是字符串长度除以2
        byte[] arrOut = new byte[iLen / 2];
        for (int i = 0; i < iLen; i = i + 2) {
            String strTmp = new String(arrB, i, 2);
            arrOut[i / 2] = (byte) Integer.parseInt(strTmp, 16);
        }
        return arrOut;
    }
    /**
     * 加密
     * @param pass
     * @return
     */
    public static String encode(String pass){
        return byteToHexString(encrypt(pass.getBytes(), key.getBytes()));
    }

    /**
     * 解密
     * @param passcode
     * @return
     */
    public static String decode(String passcode){
        return byteToHexString(decrypt(hexToByteArr(passcode), key.getBytes()));
    }

    public static void main(String[] args) {
        String content = "password111111111111111";

        System.out.println("加密前 "+ byteToHexString(content.getBytes()));
        byte[] encrypted = encrypt(content.getBytes(), key.getBytes());
        System.out.println("加密后："+ byteToHexString(encrypted));

        byte[] decrypted=decrypt(encrypted, key.getBytes());
        System.out.println("解密后："+ byteToHexString(decrypted));


        System.out.println(encode(content));
        String s = new
String(hexStringToByte(decode("159CF72C0BD2A8183D536215768C2E91556D77642F214E34"
)));
        System.out.println(s);
    }
}
```

4.2.4 Mapper实现

创建类com.heima.mappers.app.ApUserMapper

定义按照phone查询用户信息方法：

```
/**
 * 通过手机号查询用户
 * @param phone
 * @return
 */
ApUser selectByApPhone(String phone);
```

ApUserMapper.xml

```
<select id="selectByApPhone" resultMap="BaseResultMap">
    select
    <include refid="Base_Column_List" />
    from ap_user where phone = #{phone} limit 1;
</select>
```

4.2.5 代码实现

（1）检验密码service

创建接口：com.heima.login.service.ValidateService

```
/**
 * 对称加密算法 DES AES
 * 散列算法 MD5 扩展加盐 salt
 */
public interface ValidateService {

    /**
     * DES验证
     * @param user
     * @param db
     * @return
     */
    boolean validDES(ApUser user, ApUser db);

    /**
     * MD5验证
     * @param user
     * @param db
     * @return
     */
    boolean validMD5(ApUser user, ApUser db);

    /**
     * MD5加盐验证
     * @param user
     * @param db
     * @return
     */
    boolean validMD5WithSalt(ApUser user, ApUser db);
}
```

实现类：

创建类：com.heima.login.service.impl.ValidateServiceImpl

```
@Service
public class ValidateServiceImpl implements ValidateService {

    @Override
    public boolean validDES(ApUser user, ApUser db) {
        if(db.getPassword().equals(DESUtils.encode(user.getPassword()))){
            return true;
        }
        return false;
    }

    @Override
    public boolean validMD5(ApUser user, ApUser db) {
        if(db.getPassword().equals(MD5Utils.encode(user.getPassword()))){
            return true;
        }
        return false;
    }

    @Override
    public boolean validMD5WithSalt(ApUser user, ApUser db) {

 if(db.getPassword().equals(MD5Utils.encodeWithSalt(user.getPassword(),db.getSal
t()))){
            return true;
        }
        return false;
    }
}
```

（2）登录认证

修改类：com.heima.login.service.ApUserLoginService

定义登录验证接口：

```
/**
 * 用户登录验证V2
 * @param user
 * @return
 */
ResponseResult loginAuthV2(ApUser user);
```

ApUserLoginServiceImpl

```
@Override
public ResponseResult loginAuthV2(ApUser user) {
    //验证参数
    if(StringUtils.isEmpty(user.getPhone()) ||
StringUtils.isEmpty(user.getPassword())){
        return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_INVALID);
    }
    //查询用户
    ApUser dbUser = apUserMapper.selectByApPhone(user.getPhone());
```

```
    if(dbUser==null){
        return
ResponseResult.errorResult(AppHttpCodeEnum.AP_USER_DATA_NOT_EXIST);
    }

    //选择不同的加密算法实现
    boolean isValid = validateService.validMD5(user, dbUser);
    //        boolean isValid = validateService.validMD5WithSalt(user, dbUser);
    //        boolean isValid = validateService.validDES(user, dbUser);

    if(!isValid){
        return ResponseResult.errorResult(AppHttpCodeEnum.LOGIN_PASSWORD_ERROR);
    }
    //登录处理
    //设置redis
    redisTemplate.opsForValue().set("ap-user-"+user.getId(),
JSON.toJSONString(user), USER_EXPIRE);
    //登录成功发送消息
    kafkaSender.sendUserLoginMessage(user);
    dbUser.setPassword("");
    Map<String,Object> map = Maps.newHashMap();
    map.put("token", AppJwtUtil.getToken(dbUser));
    map.put("user",dbUser);
    return ResponseResult.okResult(map);
}
```

LoginControllerApi

创建类：com.heima.login.apis.ArticleSearchControllerApi

```
public interface LoginControllerApi {

    /**
     * 用户登录验证
     * @param user
     * @return
     */
    public ResponseResult login(ApUser user);
}
```

创建类：com.heima.login.controller.v2.LoginController

```
@RestController
@RequestMapping("/api/v2/login")
public class LoginV2Controller implements LoginControllerApi {

    @Autowired
    private ApUserLoginService apUserLoginService;

    @PostMapping("login_auth")
    @Override
    public ResponseResult login(@RequestBody ApUser user) {
        return apUserLoginService.loginAuthV2(user);
    }
}
```