第一章 项目介绍和工程搭建

学习目标

- 熟悉移动端应用系统的架构设计
- 熟悉大型软件系统设计中的各种图形结构
- 熟悉数据库分库分表设计技巧
- 熟悉Spring boot2.0+JavaConfig项目封装配置方式
- 完成文章列表的后台开发

1. 项目介绍

1.1项目背景

随着智能手机的普及，人们更加习惯于通过手机来看新闻。由于生活节奏的加快，很多人只能利用碎片时间来获取信息，因此，对于移动资讯客户端的需求也越来越高。黑马头条项目正是在这样背景下开发出来。黑马头条项目采用当下火热的微服务+大数据技术架构实现。本项目主要着手于获取最新最热新闻资讯，通过大数据分析用户喜好精确推送咨询新闻

1.2 项目概述

黑马头条项目是对在线教育平台业务进行大数据统计分析的系统。碎片化、切换频繁、社交化和个性化现如今成为人们阅读行为的标签。黑马头条对海量信息进行搜集，通过系统计算分类，分析用户的兴趣进行推送从而满足用户的需求。

1.3 项目术语定义

- 项目：泛指黑马头条整个项目或某一项目模块
- 工程：泛指黑马头条某一项目的源码工程

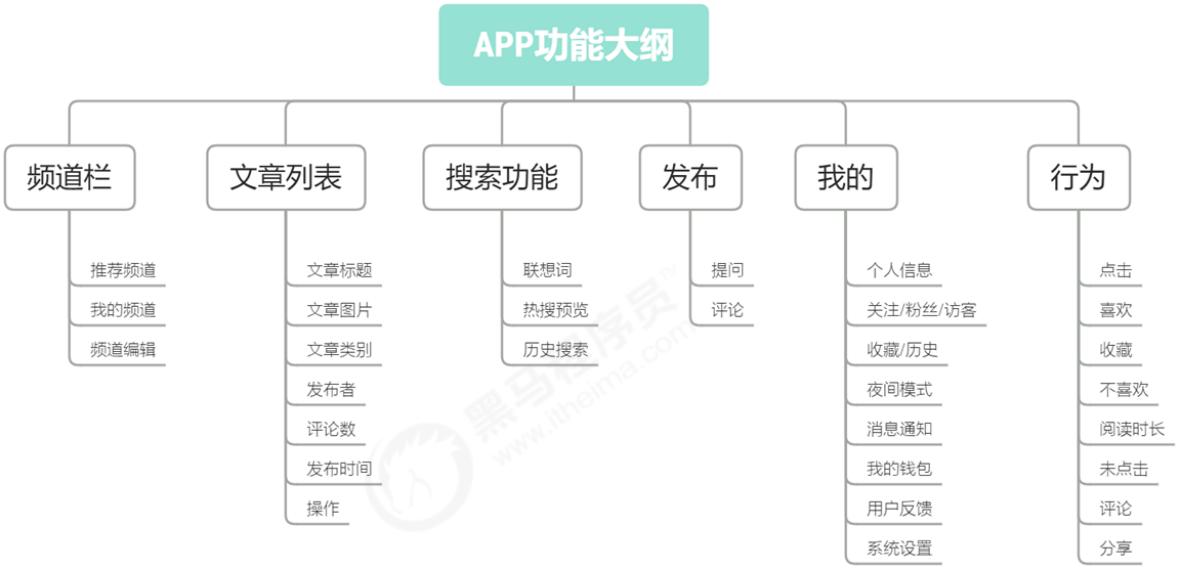
- 用户：泛指黑马头条APP用户端用户
- 自媒体人：泛指通过黑马自媒体系统发送文章的用户

- App：泛指黑马头条APP
- WeMedia：泛指黑马头条自媒体系统
- Admin：泛指黑马头条管理系统

2. 项目需求

功能需求项目模块结构

2.1 APP主要功能大纲



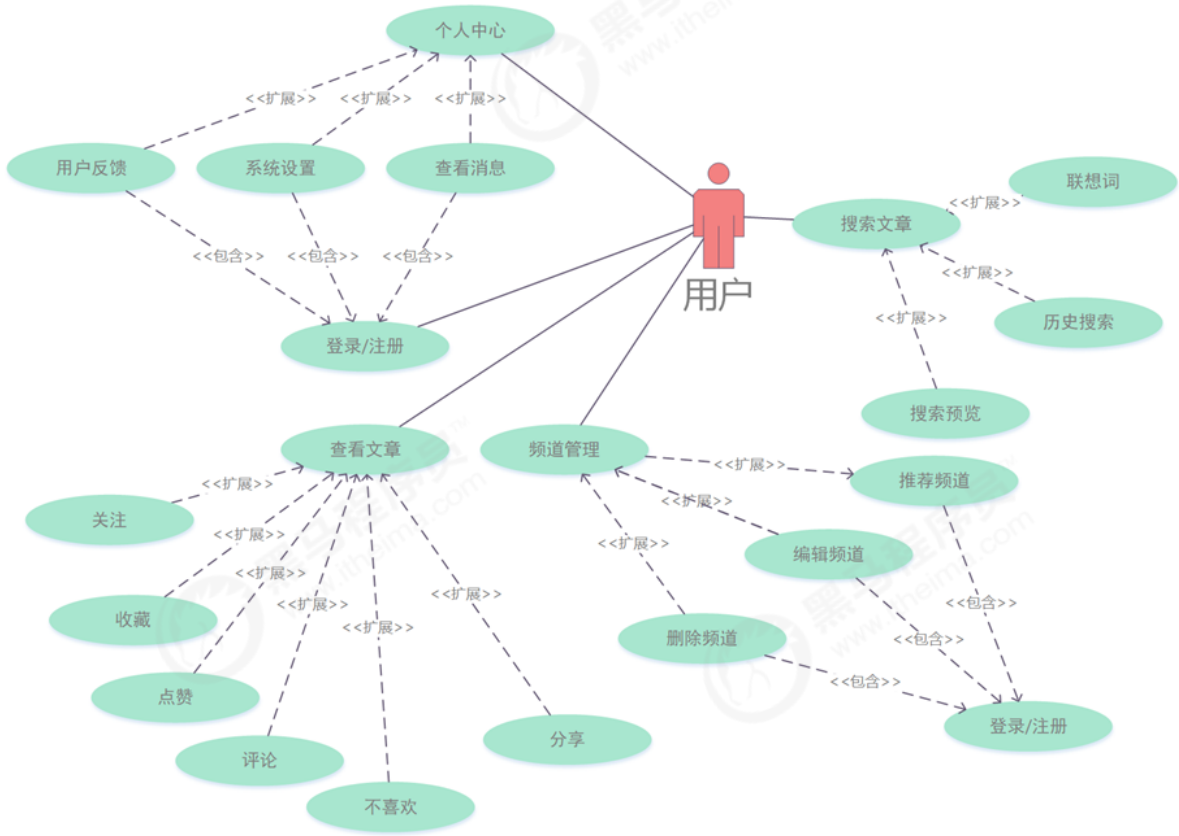
- 频道栏：用户可以通过此功能添加自己感兴趣的频道，在添加标签时，系统可依据用户喜好进行推荐
- 文章列表：需要显示文章标题、文章图片、评论数等信息，且需要监控文章是否在APP端展现的行为

- 搜索文章：联想用户想搜索的内容，并记录用户的历史搜索信息

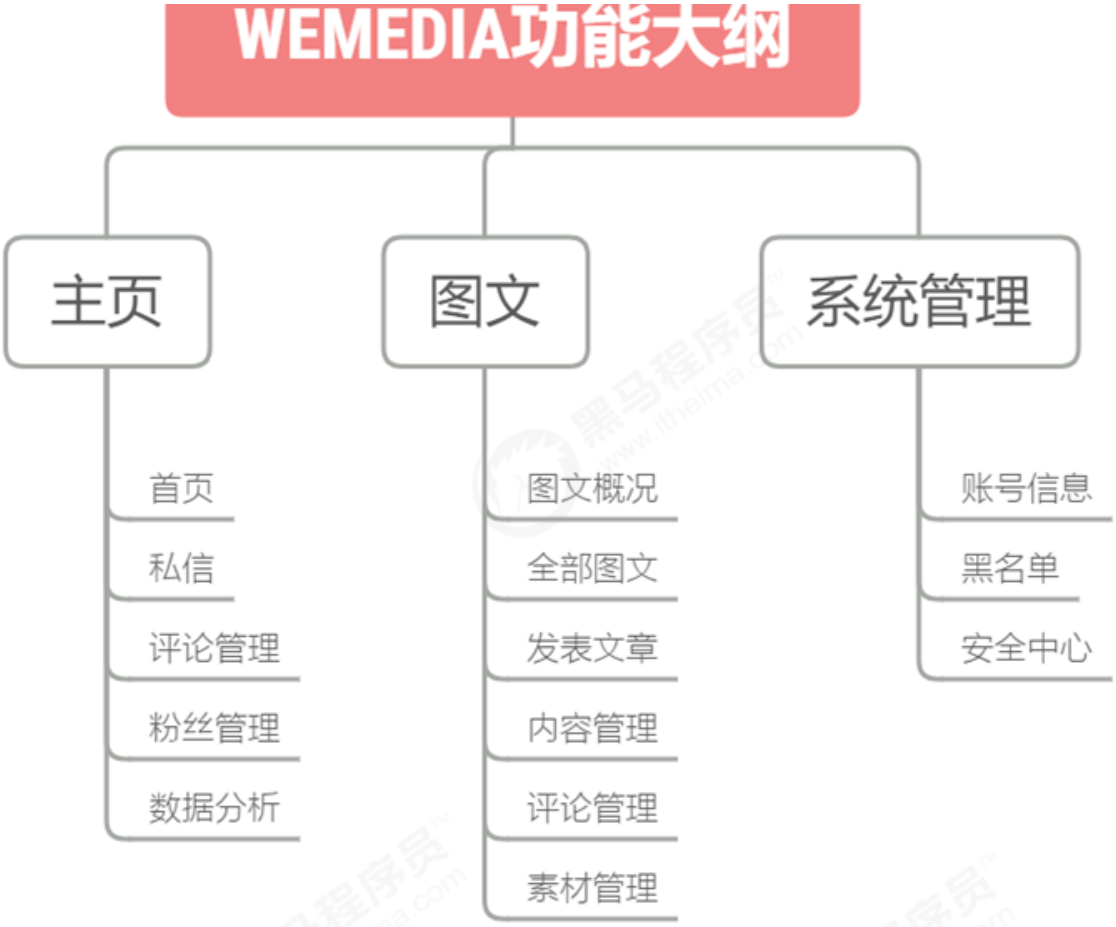
作；除此之外还需要收集用户查看文章的时间，是否看我等行为信息

- 实名认证：用户可以进行身份证认证和实名认证，实名认证之后即可成为自媒体人，在平台上发布文章
- 注册登录：登录时，验证内容为手机号登录/注册，通过手机号验证码进行登录/注册，首次登录用户自动注册账号。

2.2 APP用例图（主要功能）



2.3 WEMEDIA功能大纲

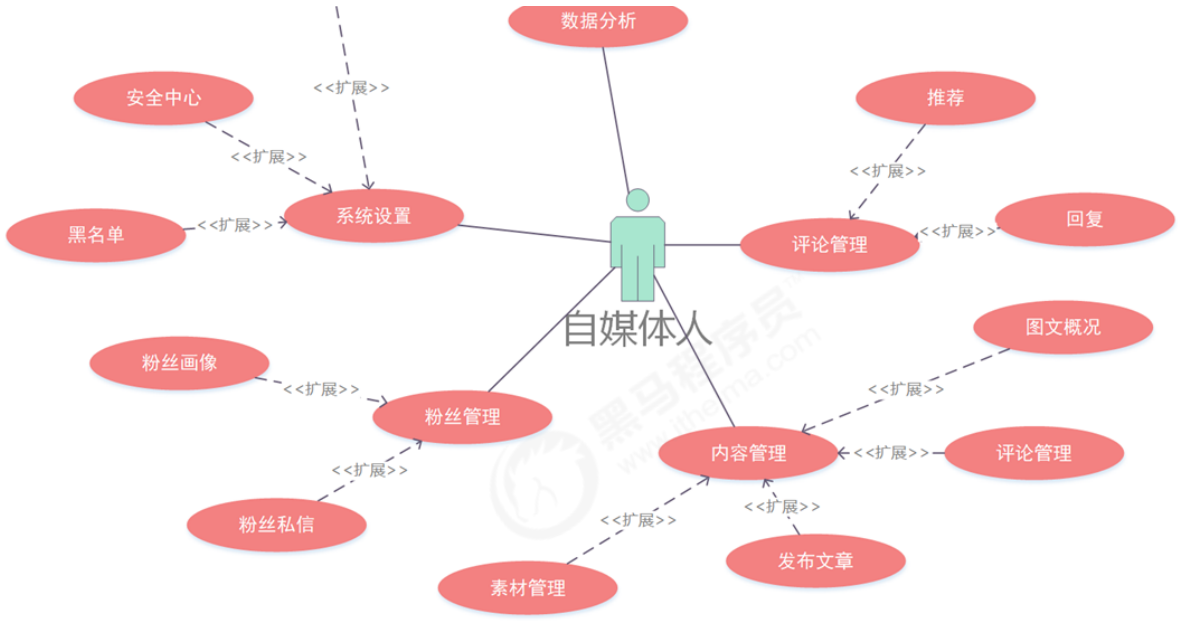


- 内容管理：自媒体用户管理文章页面，可以根据条件进行筛选，文章包含草稿、已发布、未通过、已撤回状态。用户可以对文章进行修改，上/下架操作、查看文章状态等操作
- 评论管理：管理文章评论页面，显示用户已发布的全部文章，可以查看文章总评论数和粉丝评论数，可以对文章进行关闭评论等操作

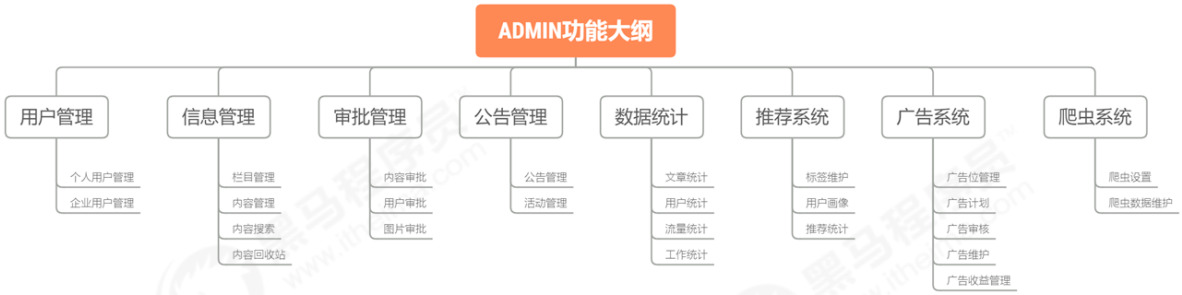
- 素材管理：管理自媒体文章发布的图片，便于用户发布带有多张图片的文章

- 图文数据：自媒体人发布文章的数据：阅读数、评论数、收藏了、转发量，用户可以查看对应文章的阅读数据
- 粉丝画像：内容包括：粉丝性别分布、粉丝年龄分布、粉丝终端分布、粉丝喜欢分类分布

2.4 WEMEDIA用例图（主要功能）



2.5 ADMIN功能大纲



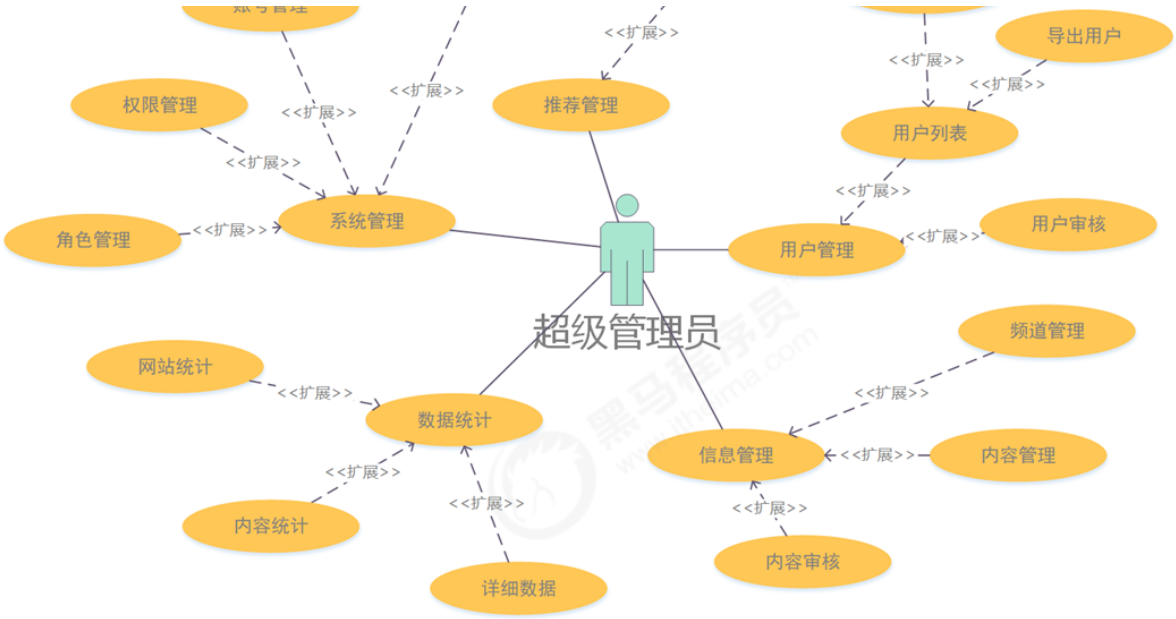
- 用户管理：系统后台用来维护用户信息，可以对用户进行增删改查操作，对于违规用户可以进行冻结操
- 用户审核：管理员审核用户信息页面，用户审核分为身份审核和实名审核，身份审核是对用户的身份信息进行审核，包括但不限于工作信息、资质信息、经历信息等；实名认证是对用户实名身份进行认证

- 内容管理：管理员查询现有文章，并对文章进行新增、删除、修改、置顶等操作
- 内容审核：管理员审核自媒体人发布的内容，包括但不限于文章文字、图片、敏感信息等
- 频道管理：管理频道分类界面，可以新增频道，查看频道，新增或修改频道关联的标签

- 网站统计：统计内容包括：日活用户、访问量、新增用户、访问量趋势、热门搜索、用户地区分布等数据
- 内容统计：统计内容包括：文章采集量、发布量、阅读量、阅读时间、评论量、转发量、图片量等数据

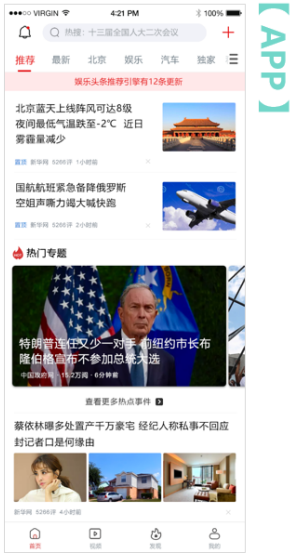
- 权限管理：超级管理员对后台管理员账号进行新增或删除角色操作

2.6 ADMIN用例图（主要功能）



2.7 其它需求

2.8 交互需求

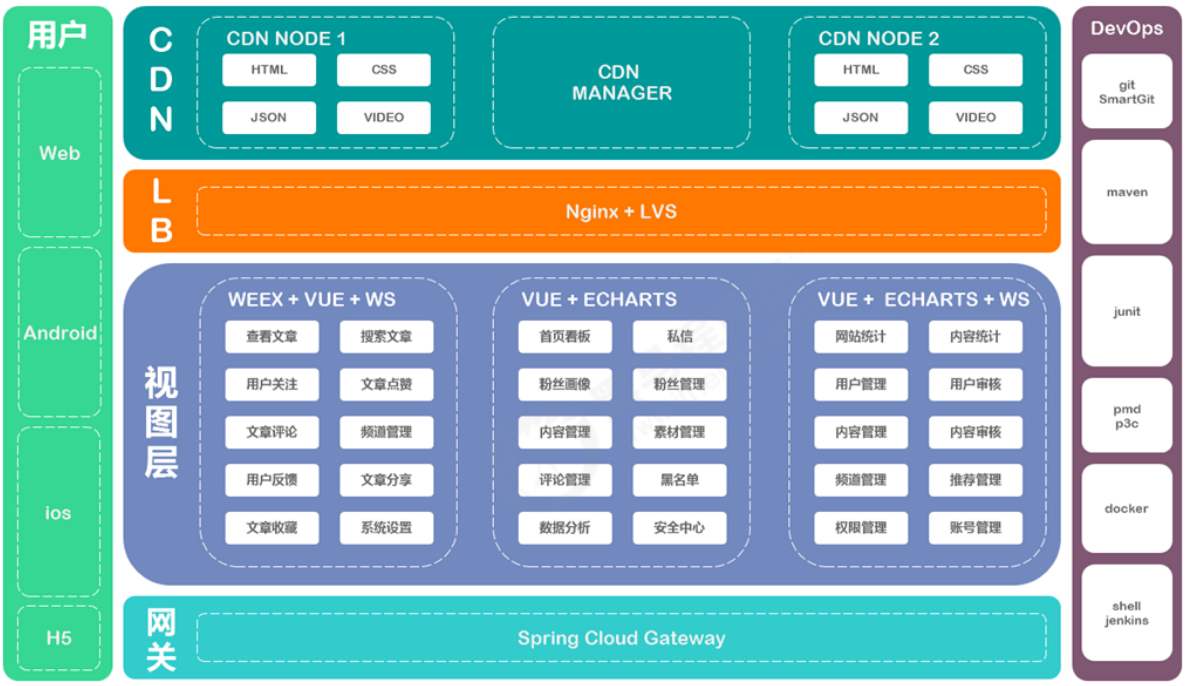


更多原型详见资料下文件：

黑马头条-原型.zip

3. 项目技术介绍

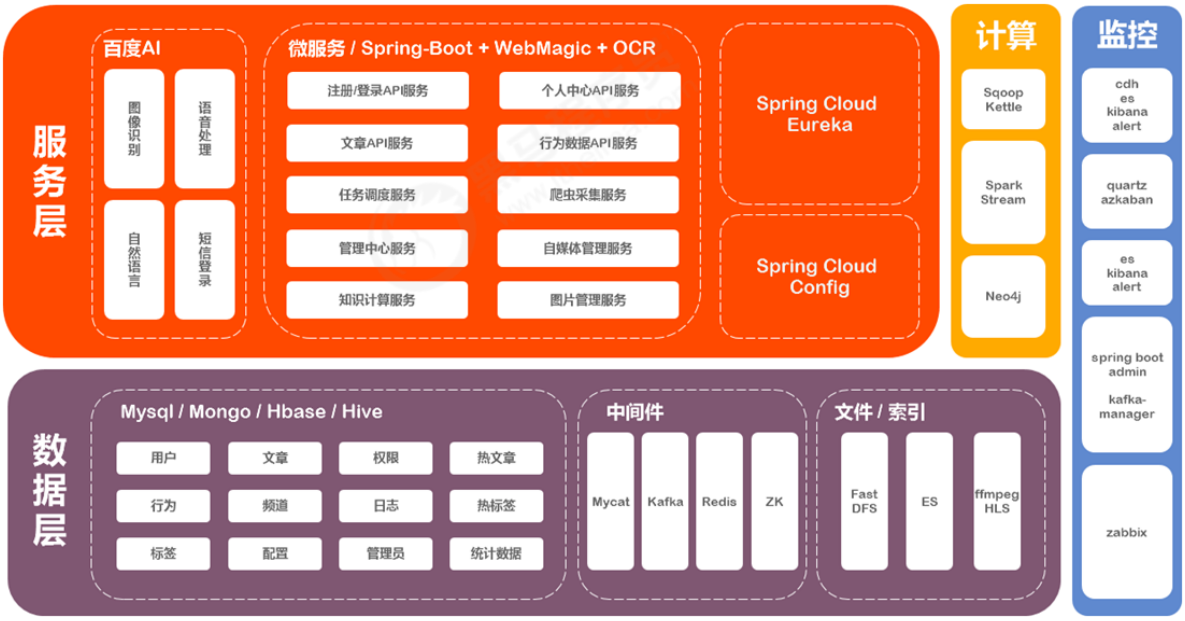
3.1 技术栈-基础六层技术



- Weex+Vue+WebSocket ：使用Weex跨平台开发工具，整合集成VUE框架，完成黑马头条移动端功能开发，并集成WebSocket实现即时消息（文章推荐、私信）的推送
- Vue+Echarts ： 自媒体系统使用Vue开发关键，集成Echarts图表框架，完成相关粉丝画像、数据分析等功能
- Vue+Echarts+WebSocket ： 管理系统也是使用Vue开发，集成Echarts，完成网站统计、内容统计等功能，集成WebSocket，实现系统看板实时数据自动化更新
- Spring-Cloud-Gateway : 微服务之前架设的网关服务，实现服务注册中的API请求路由，以及控制流速控制和熔断处理都是常用的架构手段，而这些功能Gateway天然支持
- PMD&P3C : 静态代码扫描工具，在项目中扫描项目代码，检查异常点、优化点、代码规范等，为开发团队提供规范统一，提升项目代码质量
- Junit : 在持续集成思想中，单元测试偏向自动化过程，项目通过Junit+Maven的集成实现这种过程

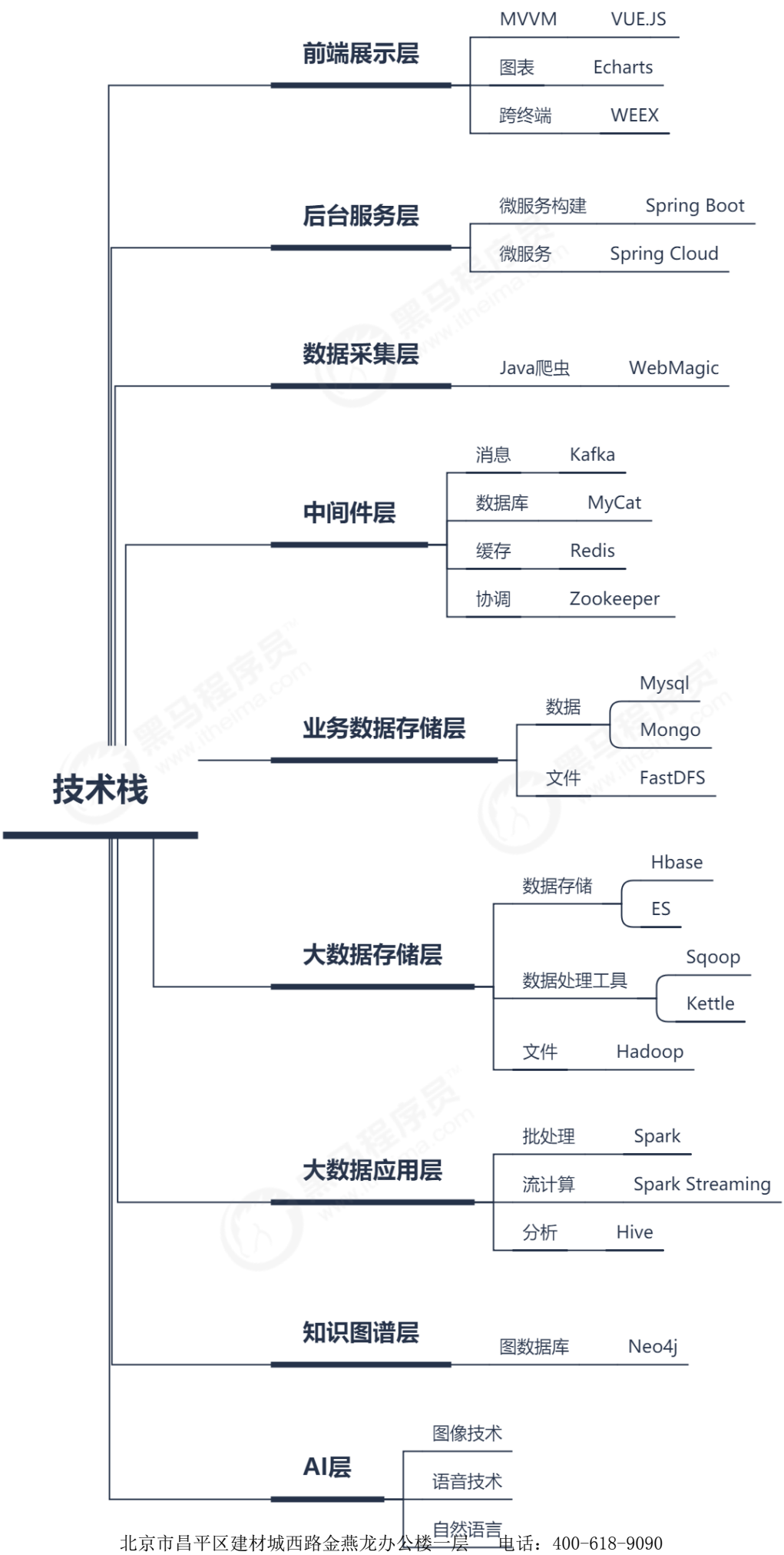
3.2 技术栈-服务四层技术

服务四层中包括中间件（Kafka、Mycat）、计算（Spark、Neo4j、Hive）、索引、微服务、大数据存储等重难点技术



- 运用WebMagic爬虫技术，完善系统内容自动化采集
- 运用Kafka完成内部系统消息通知；与客户端系统消息通知；以及实时数据计算
- 运用MyCat数据库中间件计算，对系统数据进行分开分表，提升系统数据层性能
- 运用Redis缓存技术，实现热数据的计算，NoSession等功能，提升系统性能指标
- 运用Zoookeeper技术，完成大数据节点之后的协调与管理，提升系统存储层高可用
- 使用Mysql存储用户数据，以保证上层数据查询的高性能
- 使用Mongo存储用户热数据，以保证用户热数据高扩展和高性能指标
- 使用FastDFS作为静态资源存储器，在其上实现热静态资源缓存、淘汰等功能
- 运用Habse技术，存储系统中的冷数据，保证系统数据的可靠性
- 运用ES搜索技术，对冷数据、文章数据建立索引，以保证冷数据、文章查询性能
- 运用Sqoop、Kettle等工具，实现大数据的离线入仓；或者数据备份到Hadoop
- 运用Spark+Hive进行离线数据分析，实现系统中各类统计报表
- 运用Spark Streaming + Hive+Kafka实现实时数据分析与应用；比如文章推荐
- 运用Neo4j知识图谱技术，分析数据关系，产出知识结果，并应用到上层业务中，以帮助用户、自媒体、运营效果/能力提升。比如粉丝等级计算
- 运用AI技术，来完成系统自动化功能，以提升效率及节省成本。比如实名认证自动化

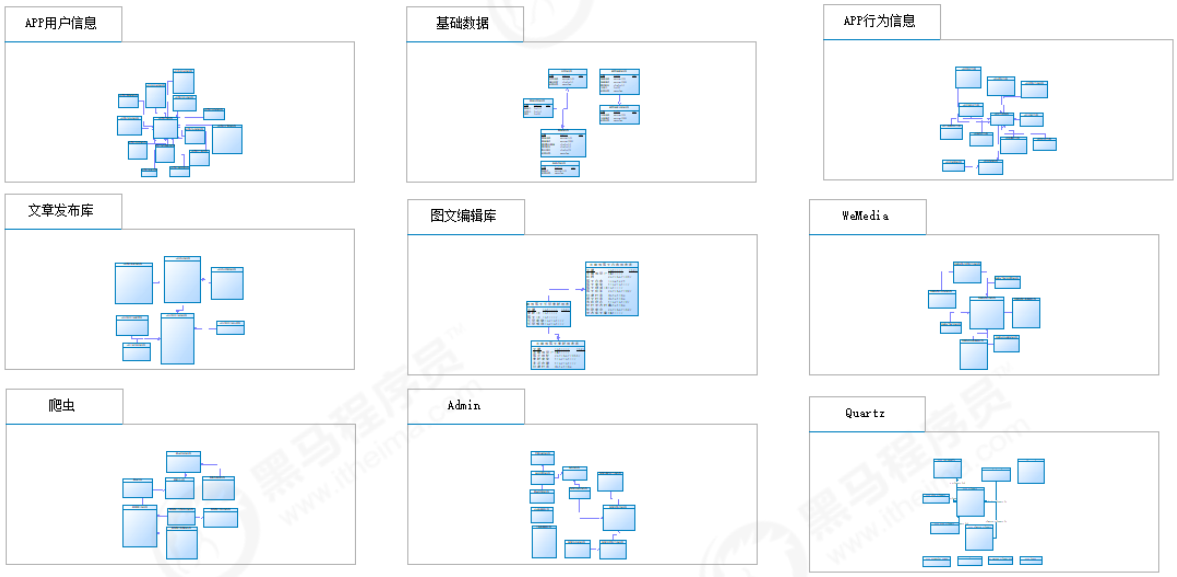
3.1 技术栈-分布


技术栈
前端展示层
MVVM VUE.JS
图表 Echarts
跨终端 WEEX
后台服务层
微服务构建 Spring Boot
微服务 Spring Cloud
数据采集层
Java爬虫 WebMagic
中间件层
消息 Kafka
数据库 MyCat
缓存 Redis
协调 Zookeeper
业务数据存储层
数据 Mysql Mongo
文件 FastDFS
大数据存储层
数据存储 Hbase ES
数据处理工具 Sqoop Kettle
文件 Hadoop
大数据应用层
批处理 Spark
流计算 Spark Streaming
分析 Hive
知识图谱层
图数据库 Neo4j
AI层
图像技术
语音技术
自然语言

- 【分层】：项目技术按分层分类，涉及前端、后台、数据采集、中间件、业务数据存储、大数据存储、大数据应用、知识图谱、AI等9个层面的技术
- 【领域】：项目技术按领域分类，涉及MVVM、图表、跨终端、微服务、消息中间件、数据库中间件、爬虫、大数据存储、大数据流计算、大数据分析、知识图谱等22个领域的主流技术
- 【技术】：项目共涉及22个主要技术框架的综合运用

4. 数据库设计

4.1 ER图设计



er图设计划分出了9个库，各个库主要解决的是某一个特定的业务。

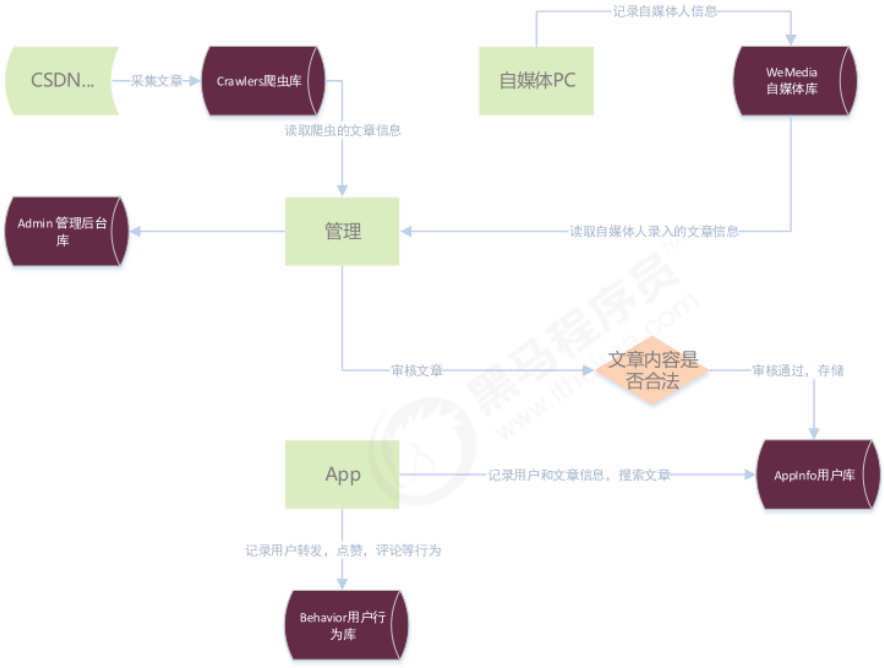
数据库设计规范，详见资料文件夹下《黑马头条-数据库规范设计说明书.md》文件。

PowerDesinger工具使用，详见资料文件夹下'powerdesinger的基本使用'文件夹里的《powerdesinger的基本使用》文件。

4.2 分库设计

黑马头条项目采用的分库分表设计，因为业务比较复杂，后期的访问量巨大，为了分摊数据库的压力，整个项目用的不只是一个数据库。其中核心库有**5**个，每一个数据库解决的是一个业务点，非常接近与实际项目设计。

- AppInfo app信息库，主要存储用户信息，文章信息，用户动态，用户评论，用户认证等信息
- Behavior 用户行为库，主要存储用户行为，包括用户的转发，点赞，评论行为等
- WeMedia 多媒体库，主要存储多媒体人图文数据统计，账号信息，粉丝相关信息等。
- Crawlers 爬虫库，主要存储从网络上爬取的文章信息等。
- Admin 后台管理库，主要存储后台管理员的信息。



黑马项目中的文章采用了多库设计的方式，以减少高并发情况下核心数据库表压力，共计设计为三个库表：

- ap_article：APP用户读取文章数据和记录次数
- cl_news：爬虫爬得文章数据
- wm_news：自媒体用户发布的文章数据

cl_news和wm_news中的数据审核通过之后发布到ap_article中。

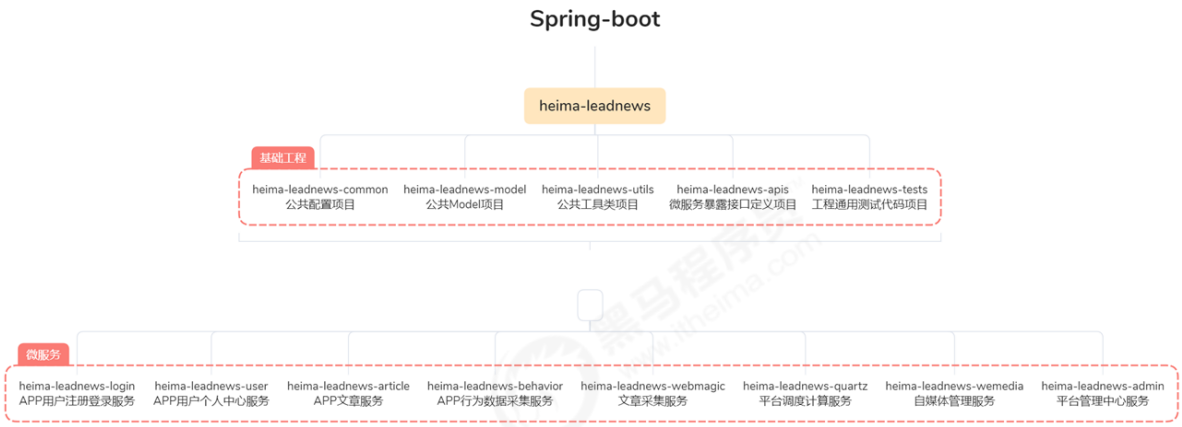
4.4 冗余设计

黑马头条项目全部采用逻辑关联，没有采用主外键约束。也是方便数据源冗余，尽可能少的使用多表关联查询。冗余是为了效率，减少join。单表查询比关联查询速度要快。某个访问频繁的字段可以冗余存放在两张表里，不用关联了。

如查询一个订单表需要查询该条订单的用户名称，就必须join另外用户表，如果业务表很大，那么就会查询的很慢，这个时候我们就可以使用冗余来解决这个问题，在新建订单的同时不仅仅需要把用户ID存储，同时也需要存储用户的名称，这样我们在查询订单表的时候就不需要去join另外用户表，也能查询出该条订单的用户名称。这样的冗余可以直接的提高查询效率，单表更快。

4.5 导入数据库

5. 后端工程结构



5.1 后端工程说明

后端工程基于Spring-boot 2.1.5.RELEASE 版本构建，工程父项目为heima-leadnews，并通过继承方式集成Spring-boot。

【父项目下分4个公共子项目】：

- heima-leadnews-common : 是整个工程的配置核心，包括所有集成三方框架的配置定义，比如redis、kafka等。除此之外还包括项目每个模块及整个项目的常量定义;
- heima-leadnews-model ：项目中用到的Dto、Pojo、Mapper、Enums定义工程;
- heima-leadnews-utils : 工程公用工具类项目，包含加密/解密、Date、JSON等工具类;
- heima-leadnew-apis : 整个项目微服务暴露的接口的定义项目，按每个模块进行子包拆分;

【多个微服务】：

- heima-leadnews-login：用于实现APP+自媒体端用户的登录与注册功能；
- heima-leadnews-user：用于实现APP端用户中心的功能，比如我的收藏、我的粉丝等功能；
- heima-leadnews-article：用于实现APP端文章的获取与搜索等功能；还包括频道、标签等功能；
- heima-leadnews-behavior：用于实现APP端各类行为数据的上传服务；
- heima-leadnews-webmagic：用于实现文章数据的自动化爬取功能；
- heima-leadnews-quartz：用于封装项目中所有的调度计算任务；
- heima-leadnews-wemedia：用于实现自媒体管理端的功能；
- heima-leadnews-admin：用于实现后台管理系统的功能；
- service-gateway：spring cloud 网关
- service-eureka：spring cloud 注册中心

5.2 后端通用工程搭建

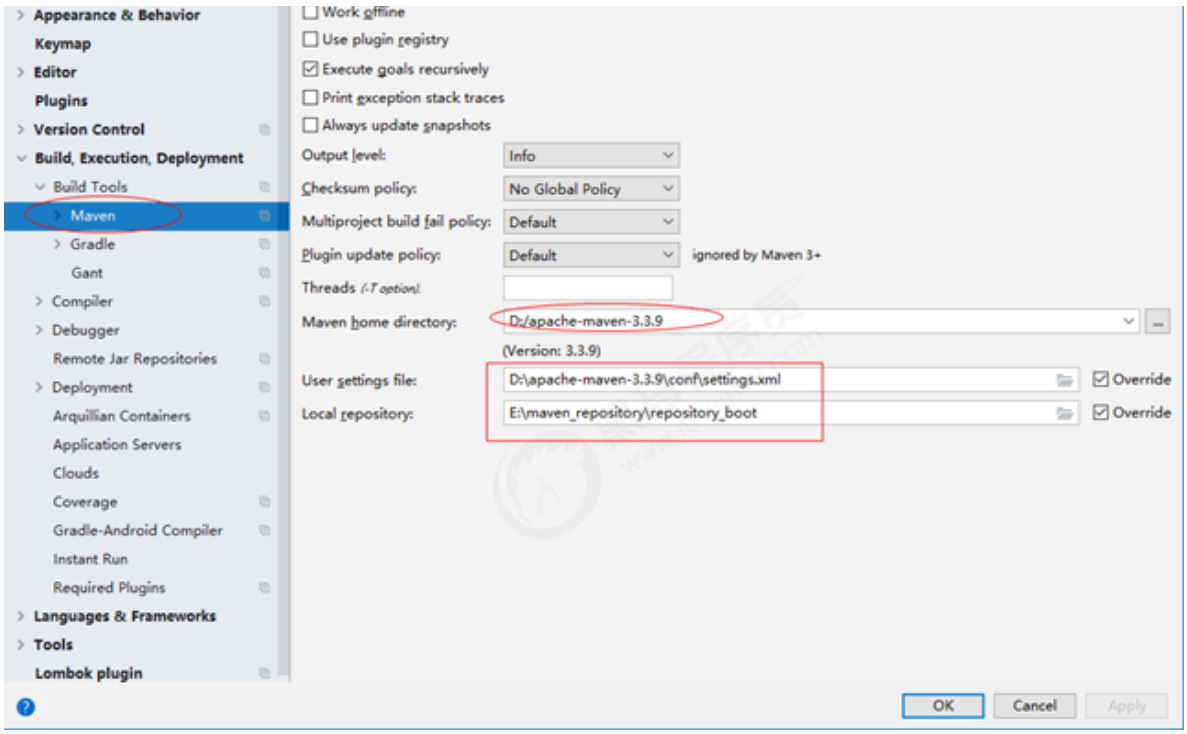
5.2.1 开发环境说明

项目依赖环境（需提前安装好）：

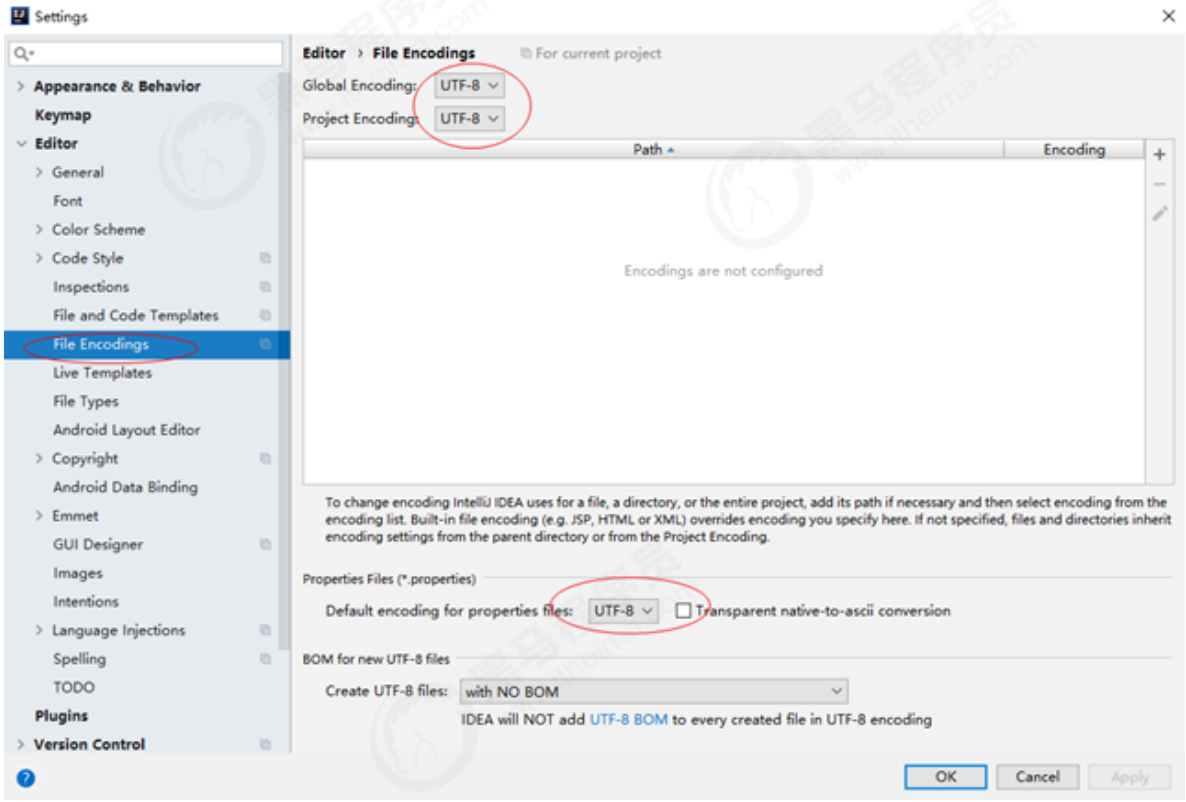
- JDK1.8
- Intellij Idea
- Tomcat 8.5
- Git

5.2.2 IDEA开发工具配置

- 设置本地仓库，建议使用资料中提供好的仓库

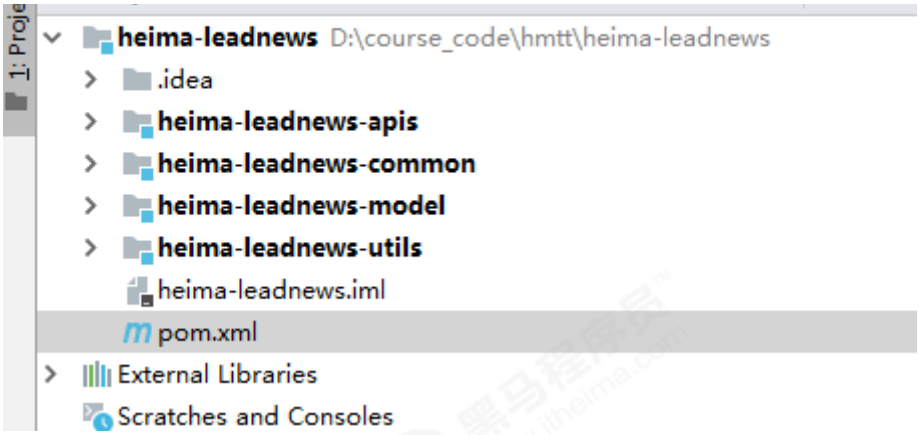


- 设置项目编码格式



5.2.3 后端初始项目导入

在当天资料中解压heima-leadnews.zip文件，拷贝到一个没有中文和空格的目录，使用idea打开即可



6 后端开发-通用说明及开发规范

6.1 后端接口开发规范

6.1.1 开发原则

- 自顶向下的设计原则：功能应该从表现层分析再到控制层、服务层、持久层逐层设计
- 自底向上的开发原则：上层需调用下层，因此开发应从底层向上层逐层开发

  项目中开发的层次次序参考DB->中间件->持久层->服务层->控制层
- 单一职责的开发原则：类或者方法提供的功能应该单一明确，特别越底层越应单一职责，以便维护

  项目中Mapper方法必须功能单一，参数明确，拒绝两种以上的持久逻辑使用同一个Mapper方法
- 依赖倒置的开发原则：上层依赖下层，是依赖下层接口，并不是依赖下层的实现

  项目中每层都是通过接口调用Controller->Service->Mapper

6.1.2 开发步骤

- 明确类定义：明确哪些是重用类，哪些是需要新增的类
- 明确主键规则：确认操作表的ID生成规则，是Mycat主键，还是Zk主键
- Mapper实现：查、改、删时注意是否使用mycat注解确认DN，插入时是否要插入主键id
- Service实现：可用通过时序图帮助我们梳理实现逻辑
- ControllerApi定义
- Controller实现：简单的Service层调用
- 单元测试

6.1.2 接口版本规范说明

随着业务的复杂，同一个接口可能出现多个版本，为了方便后期切换和AB测试，需要定义接口的版本号

- 在某一个微服务下访问controller的时候在包名下加一个版本号,如下

```
com.heima.article.controller.v1
```

- 在访问具体的接口方法的url映射的时候也应该加上版本说明，如下：

```
@RequestMapping("/api/v1/article")
```

ID混淆	请求和响应的连续增长的ID需要经过混淆加密
Date数化	请求和响应的的时间字段，统一转换成13位时间戳
字符编码	请求和响应的内容字符集为UTF-8
支持多格式	响应结果支持JSON和XML，可通过Header Accept设置
URL格式	Url为全小写字符，多个单词用下划线分隔
token	请求头中存放当前用户的请求token（JWT格式）
t	请求头中存放当前请求的时间，用于基本的请求时效判断
md	请求头中存放当前请求的参数验签字符串（查询串排序MD5加密）
响应格式	响应格式只接受ResponseResult，code码需定义在AppHttpCodeEnum

6.2 工具类说明

- IdsUtils工具类

  把数字类型的id做aes加密混淆，比如：在url传递的过程中，自增的id会做混淆处理
- UrlSignUtils工具类

  url签名工具类
- AppJwtUtil

  jwt字符串生成验证工具类
- AppThreadLocalUtils

  当前请求用户信息操作类

6.3 接口通用请求和响应

dto(Data Transfer Object):数据传输对象,用于展示层与服务层之间的数据传输对象

6.3.1 通用的响应对象：

不分页：com.heima.model.common.dtos.ResponseResult

```
/**
 * 通用的结果返回类
 * @param <T>
 */
public class ResponseResult<T> implements Serializable {

    private String host;

    private Integer code;

    private String errorMessage;

    private T data;

    public ResponseResult() {
```

```
    public ResponseResult(Integer code, T data) {
        this.code = code;
        this.data = data;
    }

    public ResponseResult(Integer code, String msg, T data) {
        this.code = code;
        this.errorMessage = msg;
        this.data = data;
    }

    public ResponseResult(Integer code, String msg) {
        this.code = code;
        this.errorMessage = msg;
    }

    public static ResponseResult errorResult(int code, String msg) {
        ResponseResult result = new ResponseResult();
        return result.error(code, msg);
    }

    public static ResponseResult okResult(int code, String msg) {
        ResponseResult result = new ResponseResult();
        return result.ok(code, null, msg);
    }

    public static ResponseResult okResult(Object data) {
        ResponseResult result =
setAppHttpCodeEnum(AppHttpCodeEnum.SUCCESS,AppHttpCodeEnum.SUCCESS.getErrorMessa
ge());
        if(data!=null) {
            result.setData(data);
        }
        return result;
    }

    public static ResponseResult errorResult(AppHttpCodeEnum enums){
        return setAppHttpCodeEnum(enums,enums.getErrorMessage());
    }

    public static ResponseResult errorResult(AppHttpCodeEnum enums,String
errorMessage){
        return setAppHttpCodeEnum(enums,errorMessage);
    }

    public static ResponseResult setAppHttpCodeEnum(AppHttpCodeEnum enums){
        return okResult(enums.getCode(),enums.getErrorMessage());
    }

    private static ResponseResult setAppHttpCodeEnum(AppHttpCodeEnum
enums,String errorMessage){
        return okResult(enums.getCode(),errorMessage);
    }

    public ResponseResult<?> error(Integer code, String msg) {
```

```
        return this;
    }

    public ResponseResult<?> ok(Integer code, T data) {
        this.code = code;
        this.data = data;
        return this;
    }

    public ResponseResult<?> ok(Integer code, T data, String msg) {
        this.code = code;
        this.data = data;
        this.errorMessage = msg;
        return this;
    }

    public ResponseResult<?> ok(T data) {
        this.data = data;
        return this;
    }

    public Integer getCode() {
        return code;
    }

    public void setCode(Integer code) {
        this.code = code;
    }

    public String getErrorMessage() {
        return errorMessage;
    }

    public void setErrorMessage(String errorMessage) {
        this.errorMessage = errorMessage;
    }

    public T getData() {
        return data;
    }

    public void setData(T data) {
        this.data = data;
    }

    public String getHost() {
        return host;
    }

    public void setHost(String host) {
        this.host = host;
    }
}
```

分页通用返回：com.heima.model.common.dtos.PageResponseResult

```
    private Integer size;
    private Integer total;

    public PageResponseResult(Integer currentPage, Integer size, Integer total)
{
        this.currentPage = currentPage;
        this.size = size;
        this.total = total;
    }

    public int getCurrentPage() {
        return currentPage;
    }

    public void setCurrentPage(int currentPage) {
        this.currentPage = currentPage;
    }

    public int getSize() {
        return size;
    }

    public void setSize(int size) {
        this.size = size;
    }

    public int getTotal() {
        return total;
    }

    public void setTotal(int total) {
        this.total = total;
    }
}
```

6.3.2 通用的请求dtos

com.heima.model.common.dtos.PageRequestDto

```
@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);
        }
    }
}
```

6.3.3 定义异常枚举

com.heima.model.common.enums.AppHttpCodeEnum

```
public enum AppHttpCodeEnum {

    // 成功段0
    SUCCESS(0,"操作成功"),
    // 登录段1~50
    NEED_LOGIN(1,"需要登录后操作"),
    LOGIN_PASSWORD_ERROR(2,"密码错误"),
    // TOKEN50~100
    TOKEN_INVALID(50,"无效的TOKEN"),
    TOKEN_EXPIRE(51,"TOKEN已过期"),
    TOKEN_REQUIRE(52,"TOKEN是必须的"),
    // SIGN验签 100~120
    SIGN_INVALID(100,"无效的SIGN"),
    SIG_TIMEOUT(101,"SIGN已过期"),
    // 参数错误 500~1000
    PARAM_REQUIRE(500,"缺少参数"),
    PARAM_INVALID(501,"无效参数"),
    PARAM_IMAGE_FORMAT_ERROR(502,"图片格式有误"),
    SERVER_ERROR(503,"服务器内部错误"),
    // 数据错误 1000~2000
    DATA_EXIST(1000,"数据已经存在"),
    AP_USER_DATA_NOT_EXIST(1001,"ApUser数据不存在"),
    DATA_NOT_EXIST(1002,"数据不存在"),
    // 数据错误 3000~3500
    NO_OPERATOR_AUTH(3000,"无权限操作");

    int code;
    String errorMessage;

    AppHttpCodeEnum(int code, String errorMessage){
        this.code = code;
        this.errorMessage = errorMessage;
    }

    public int getCode() {
        return code;
    }

    public String getErrorMessage() {
        return errorMessage;
    }
}
```

6.4 jackson封装解决字段序列化及反序列化

主要针对一些数据进行过滤设置，使用jackson来实现，比如一些自增的id值或者一些混淆的属性，在互联网传输的过程中最好不要轻易暴露在外面，这样安全性就比较低，应该对这些自增的值进行加密混淆。

日期处理：在网络上进行传输的时候不要直接传输日期的格式，传输的格式为13位时间戳，如果在每个日期字段都手动设置的话是比较费时费力的，所以也可以做成自动转换，用的也是jackson来完成

- IdEncrypt 自定义注解 作用在需要混淆的字段属性上，用于非id的属性上 在model包下
- DateDeserializer 用于处理日期输入反序列化
- DateSerializer 用于日期的序列化输出
- ConfusionSerializer 用于序列化自增数字的混淆
- ConfusionDeserializer 用于反序列化自增数字的混淆解密
- ConfusionSerializerModifier 用于过滤序列化时处理的字段
- ConfusionDeserializerModifier 用于过滤反序列化时处理的字段
- ConfusionModule 用于注册模块和修改器
- InitJacksonConfig 提供自动化配置默认ObjectMapper，让整个框架自动处理日期和id混淆

6.5 通用环境说明

6.5.1 多环境切换

在每一个微服务的工程中的根目录下创建三个文件，方便各个环境的切换

（1）maven_dev.properties

定义开发环境的配置

（2）maven_prod.properties

定义生产环境的配置

（3）maven_test.properties

定义测试环境的配置，开发阶段使用这个测试环境

默认加载的环境为test，在打包的过程中也可以指定参数打包 package -P test/prod/dev

具体配置，请查看父工程下的maven插件的profiles配置

```
<profiles>
    <profile>
        <id>dev</id>
        <build>
            <filters>
                <filter>maven_dev.properties</filter>
            </filters>
        </build>
    </profile>
    <profile>
        <id>test</id>
        <activation>
            <activeByDefault>true</activeByDefault>
        </activation>
        <build>
            <filters>
                <filter>maven_test.properties</filter>
            </filters>
        </build>
    </profile>
    <profile>
        <id>prod</id>
        <build>
            <filters>
                <filter>maven_prod.properties</filter>
```

```
    </profile>
</profiles>
```

6.5.2 mysql的环境配置

在heima-leadnews-common设置配置文件mysql-core-jdbc.properties

```
### ======================= 核心数据库连接配置 ========================
# 数据库连接字符串
mysql.core.jdbc-url=${mysql.core.jdbc.url}
# 数据库连接名称
mysql.core.jdbc-user-name=${mysql.core.jdbc.username}
# 数据库连接密码，密码需要反转
mysql.core.jdbc-password=${mysql.core.jdbc.password}
# 数据库连接驱动
mysql.core.jdbc-driver=${mysql.core.jdbc.driver}
# mybatis mapper.xml存放在classpath下的根文件夹名称
mysql.core.root-mapper=${mysql.core.root.mapper}
# mybatis pojo对象别名扫描包
mysql.core.aliases-package=${mysql.core.aliases.package}
# 事务扫描包自动代理扫描包
mysql.core.tx-scan-package=${mysql.core.tx.scan.package}
```

这里面的内容统统都是从maven_test.properties读取

```
mysql.core.jdbc.url=jdbc:mysql://localhost:3306/heima-leadnews?
autoReconnect=true&useUnicode=true&characterEncoding=utf8&serverTimezone=Asia/Sh
anghai
mysql.core.jdbc.username=root
mysql.core.jdbc.password=toor

mysql.core.jdbc.driver=com.mysql.jdbc.Driver
mysql.core.root.mapper=mappers
mysql.core.aliases.package=com.heima.model.**
mysql.core.tx.scan.package=execution(* com.heima..service.*.*(..))
```

自动化配置核心数据库的连接配置com.heima.common.mysql.core.MysqlCoreConfig

```
/**
 * 自动化配置核心数据库的连接配置
 */
@Setter
@Getter
@Configuration
@ConfigurationProperties(prefix = "mysql.core")
@PropertySource("classpath:mysql-core-jdbc.properties")
@MapperScan(basePackages = "com.heima.model.mappers", sqlSessionFactoryRef =
"mysqlCoreSqlSessionFactory")
public class MysqlCoreConfig {
    String jdbcUrl;
    String jdbcUserName;
    String jdbcPassword;
    String jdbcDriver;
    String rootMapper;//mapper文件在classpath下存放的根路径
```

```
    /**
     * 这是最快的数据库连接池
     *
     * @return
     */
    @Bean
    public DataSource mysqlCoreDataSource() {
        HikariDataSource hikariDataSource = new HikariDataSource();
        hikariDataSource.setUsername(this.getJdbcUserName());
        hikariDataSource.setPassword(this.getRealPassword());
        hikariDataSource.setJdbcUrl(this.getJdbcUrl());
        //最大连接数
        hikariDataSource.setMaximumPoolSize(50);
        //最小连接数
        hikariDataSource.setMinimumIdle(5);
        hikariDataSource.setDriverClassName(this.getJdbcDriver());
        return hikariDataSource;
    }

    /**
     * 这是Mybatis的Session
     *
     * @return
     * @throws IOException
     */
    @Bean
    public SqlSessionFactoryBean
mysqlCoreSqlSessionFactory(@Qualifier("mysqlCoreDataSource") DataSource
mysqlCoreDataSource) throws IOException {
        PathMatchingResourcePatternResolver resolver = new
PathMatchingResourcePatternResolver();
        //创建sqlSessionFactory工厂对象
        SqlSessionFactoryBean sessionFactory = new SqlSessionFactoryBean();
        //数据源
        sessionFactory.setDataSource(mysqlCoreDataSource);
        //mapper文件的路径

 sessionFactory.setMapperLocations(resolver.getResources(this.getMapperFilePath(
)));
        //别名
        sessionFactory.setTypeAliasesPackage(this.getAliasesPackage());
        //开启自动驼峰标识转换
        org.apache.ibatis.session.Configuration mybatisConf = new
org.apache.ibatis.session.Configuration();
        mybatisConf.setMapUnderscoreToCamelCase(true);
        sessionFactory.setConfiguration(mybatisConf);
        return sessionFactory;
    }

    /**
     * 密码反转，简单示意密码在配置文件中的加密处理
     *
     * @return
     */
    public String getRealPassword() {
        return StringUtils.reverse(this.getJdbcPassword());
```

```
    /**
     * 拼接Mapper.xml文件的存放路径
     *
     * @return
     */
    public String getMapperFilePath() {
        return new
StringBuffer().append("classpath:").append(this.getRootMapper()).append("/**/*.x
ml").toString();
    }


}
```

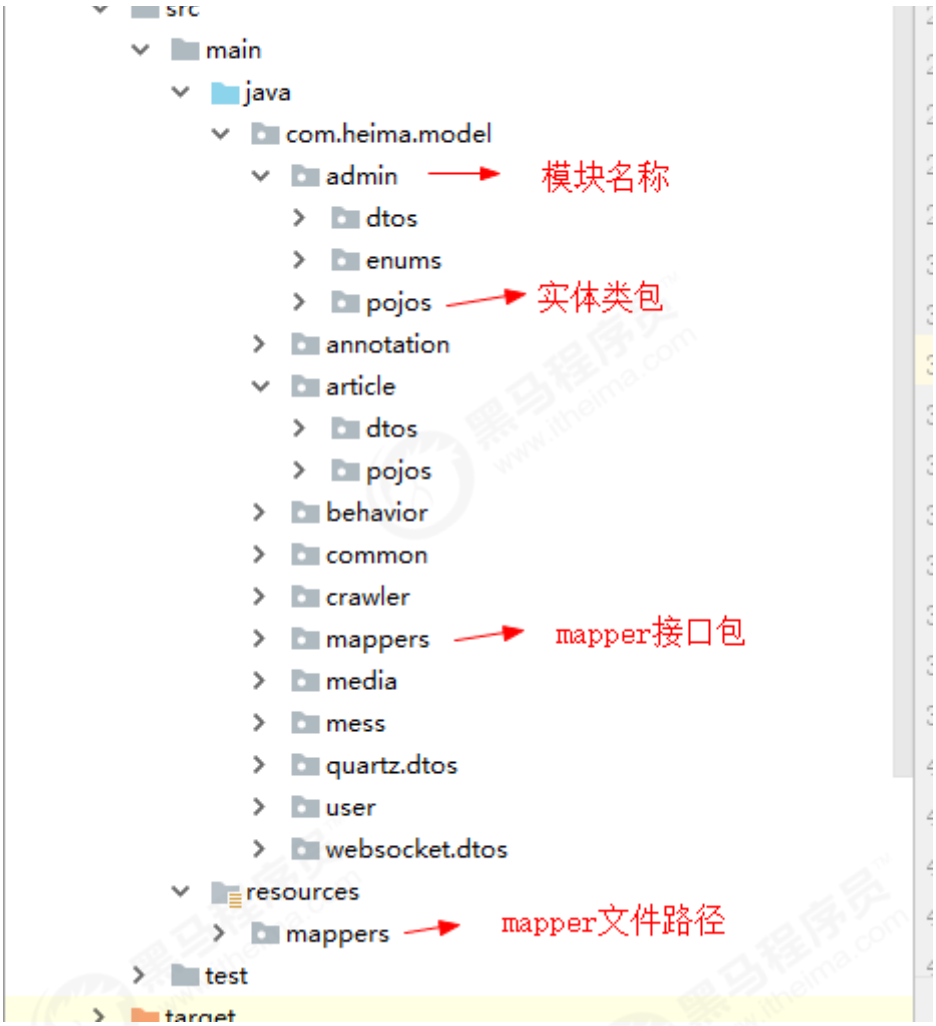
通用事务管理配置类com.heima.common.mysql.core.TransactionConfig

```
@Setter
@Getter
@Aspect
@EnableAspectJAutoProxy
@EnableTransactionManagement
@Configuration
@ConfigurationProperties(prefix="mysql.core")
@PropertySource("classpath:mysql-core-jdbc.properties")
public class TransactionConfig {

    String txScanPackage;

    /**
     * 初始化事务管理器
     * @param dataSource
     * @return
     */
    @Bean
    public DataSourceTransactionManager
mysqlCoreDataSourceTransactionManager(@Qualifier("mysqlCoreDataSource")
DataSource dataSource){
        DataSourceTransactionManager dataSourceTransactionManager = new
DataSourceTransactionManager();
        dataSourceTransactionManager.setDataSource(dataSource);
        return dataSourceTransactionManager;
    }

    /**
     * 设置事务拦截器
     * @param dataSourceTransactionManager
     * @return
     */
    @Bean
    public TransactionInterceptor
mysqlCoreDataSourceTxAdvice(@Qualifier("mysqlCoreDataSourceTransactionManager")
DataSourceTransactionManager dataSourceTransactionManager) {
        // 默认事务
        DefaultTransactionAttribute defAttr = new
DefaultTransactionAttribute(TransactionDefinition.PROPAGATION_REQUIRED);
```

```
DefaultTransactionAttribute(TransactionDefinition.PROPAGATION_REQUIRED);
        queryAttr.setReadOnly(true);
        // 设置拦截的方法
        NameMatchTransactionAttributeSource source = new
NameMatchTransactionAttributeSource();
        source.addTransactionalMethod("save*", defAttr);
        source.addTransactionalMethod("insert*", defAttr);
        source.addTransactionalMethod("delete*", defAttr);
        source.addTransactionalMethod("update*", defAttr);
        source.addTransactionalMethod("exec*", defAttr);
        source.addTransactionalMethod("set*", defAttr);
        source.addTransactionalMethod("add*", defAttr);
        source.addTransactionalMethod("get*", queryAttr);
        source.addTransactionalMethod("query*", queryAttr);
        source.addTransactionalMethod("find*", queryAttr);
        source.addTransactionalMethod("list*", queryAttr);
        source.addTransactionalMethod("count*", queryAttr);
        source.addTransactionalMethod("is*", queryAttr);

        return new TransactionInterceptor(dataSourceTransactionManager, source);
    }

    @Bean
    public Advisor txAdviceAdvisor(@Qualifier("mysqlCoreDataSourceTxAdvice")
TransactionInterceptor mysqlCoreDataSourceTxAdvice) {
        AspectJExpressionPointcut pointcut = new AspectJExpressionPointcut();
        pointcut.setExpression(txScanPackage);
        return new DefaultPointcutAdvisor(pointcut,
mysqlCoreDataSourceTxAdvice);
    }

}
```

6.4.3 实体类及mapper

所有实体类都是按业务模板划分，mapper接口单独在一个包下，如下图



com.heima.model.mappers 定义mapper接口类

resources/mapper mapper映射文件的定义

7 app端-文章列表后端开发

7.1 文章列表需求分析

文章首页为用户进入应用之后的第一个页面，在这里我我们需要根据用户是否登录的状态加载不同的文章数据；用户登录则根据其选择的频道进行加载，否则根据系统默认频道推荐,流程图如下；在首页我们加载数据之后需要考虑用户在首页的其他动作的相应，如：用户上拉刷新、用户下拉刷新、用户进入文章详情等用户动作

相关表结构分析

ap_article文章信息表

文章信息表，存储已发布的文章

id	int(11)	主键
title	varchar(50)	标题
author_id	int(11)	文章作者的ID
author_name	varchar(20)	作者昵称
channel_id	int(10)	文章所属频道ID
channel_name	varchar(10)	频道名称
layout	tinyint(1)	文章布局 0 无图文章 1 单图文章 2 多图文章
flag	tinyint(3)	文章标记 0 普通文章 1 热点文章 2 置顶文章 3 精品文章 4 大V 文章
images	varchar(1000)	文章图片 多张逗号分隔
labels	varchar(500)	文章标签最多3个 逗号分隔
likes	int(5)	点赞数量
collection	int(5)	收藏数量
comment	int(5)	评论数量
views	int(5)	阅读数量
province_id	int(11)	省市
city_id	int(11)	市区
county_id	int(11)	区县
created_time	datetime	创建时间
publish_time	datetime	发布时间
sync_status	tinyint(1)	同步状态
origin	tinyint(1)	来源

ap_user_article_list APP用户文章列表

id	int(11)	主键
user_id	int(11)	用户ID
channel_id	int(11)	频道ID
article_id	int(11)	文章ID
is_show	tinyint(1)	是否展示
recommend_time	datetime	推荐时间
is_read	tinyint(1)	是否阅读
strategy_id	int(5)	推荐算法

ap_show_behaviorAPP文章展现行为表

字段	类型	描述
id	int(11)	主键
entry_id	int(11)	实体ID
article_id	int(11)	文章ID
is_click	tinyint(1)	是否点击
show_time	datetime	文章加载时间
created_time	datetime	登录时间

ap_behavior_entry app行为实体表

APP行为实体表,一个行为实体可能是用户或者设备，或者其它

字段	类型	描述
id	int(11)	主键
type	tinyint(1)	实体类型 0终端设备 1用户
entry_id	int(11)	实体ID
created_time	datetime	创建时间
burst	varchar(40)	分片

7.2 文章列表相关接口定义

7.2.1 接口定义分析

由需求分析可知用户在首页的是可能触发的行为有加载文章列表，刷新（上拉刷新、下拉刷新）等动作，这也就是我们后端需要对应的几个数据接口，其中还包含一个隐含的用户行为接口用户记录用户是否阅读某一篇文章的行为接口，则我们可以分析出后端需要的接口有：

load接口，分两种情况，一个是登录，一个是未登录，加载多条数据（有条数的限制，size），用户可以选择频道进行数据的切换

- 登录，从后台获取用户信息，作为条件查询
- 未登录，直接加载默认数据即可。

（2）load_more接口 && load_new 接口

当用户进行刷新是，在我们的系统中定义了两种操作，第一种是上拉刷新也就是load_more，第二种是下拉刷新load_new接口，这两个接口的区别在于加载的内容的**时间不同**。

用户进入系统的时间TimeA浏览了一会首次加载的数据之后到了TimeB时间，这时候如果用户继续上拉看后面的内容则我们调用load_more接口并把TimeB时间传递到后端，后端根据TimeB时间查找当前时间之前发布的内容；

用户下拉刷新则说明用户需要当前最新的内容则调用load_new接口，将TimeA时间传到后端后端查找TimeA时间之后发布的内容。其请求参数接口设计和load接口相同。

（4）behavior 行为接口

记录用户操作行为

7.3 项目开发

在原有的工程中创建一个普通的maven工程模块，选择其作为heima-leadnews作为父工程，并给当前模块命名为heima-leadnews-article

配置我们需要的jar的maven坐标，以及我们项目中模块的项目依赖 **注意**我们的数据库相关的实体以及Mapper接口和配置文件都是存放在Model 模块。

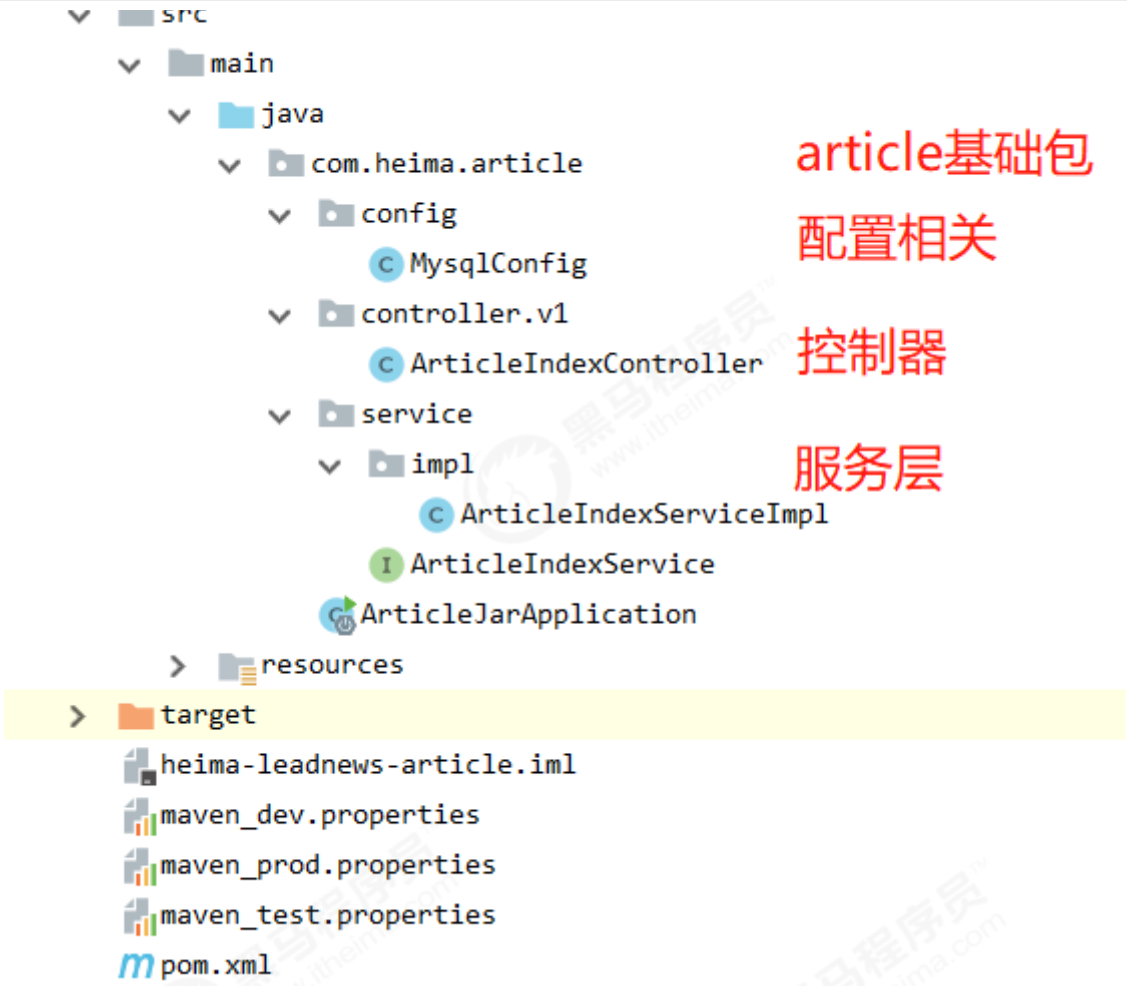
7.3.1 定义pom文件中的依赖信息

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

    <artifactId>heima-leadnews-article</artifactId>

    <dependencies>
        <!-- 引入依赖模块 -->
        <dependency>
            <groupId>com.heima</groupId>
            <artifactId>heima-leadnews-model</artifactId>
        </dependency>
        <dependency>
            <groupId>com.heima</groupId>
            <artifactId>heima-leadnews-common</artifactId>
        </dependency>
        <dependency>
```

```
        </dependency>
        <!-- Spring boot starter -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <!-- 排除默认的logback日志，使用log4j-->
            <exclusions>
                <exclusion>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-logging</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>org.slf4j</groupId>
                    <artifactId>slf4j-log4j12</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>ch.qos.logback</groupId>
                    <artifactId>logback-access</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.jsoup</groupId>
            <artifactId>jsoup</artifactId>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.dataformat</groupId>
            <artifactId>jackson-dataformat-cbor</artifactId>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.dataformat</groupId>
            <artifactId>jackson-dataformat-xml</artifactId>
        </dependency>
    </dependencies>
</project>
```

7.3.2 Article模块工程结构

注意这里可以只创建基础的包结构即可，后续会讲解每个具体的类的作用以及实现；此处可能你又疑问为什么我们的控制器加了个v1，这里我们的做法是为了兼容不同的终端版本所设立的版本号



项目根路径添加文件:maven_dev.properties

```
# log4j
log.level=DEBUG
log.pattern=%d{DEFAULT}^|%sn^|%level^|%t^|%c^|%M^|%msg%n
```

项目根路径添加文件:maven_prod.properties

```
# log4j
log.level=DEBUG
log.pattern=%d{DEFAULT}^|%sn^|%level^|%t^|%c^|%M^|%msg%n
```

项目根路径添加文件:maven_test.properties

```
#log4j
log.level=DEBUG
log.pattern=%d{DEFAULT}^|%sn^|%level^|%t^|%c^|%M^|%msg%n
```

在resource目录下创建application.properties和log4j2.xml

application.properties

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

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <properties>
        <property name="CONSOLE_PATTERN">${log.pattern}</property>
        <property name="FILE_NAME">${project.build.finalName}</property>
    </properties>
    <!--先定义所有的appender-->
    <appenders>
        <!--这个输出控制台的配置-->
        <console name="Console" target="SYSTEM_OUT">
            <!--输出日志的格式-->
            <PatternLayout pattern="${CONSOLE_PATTERN}"/>
        </console>
        <!-- 这个会打印出所有的info及以下级别的信息，每次大小超过size，则这size大小的日志会自动存入按年份-月份建立的文件夹下面并进行压缩，作为存档-->
        <RollingFile name="RollingFileInfo" fileName="${sys:user.home}/logs/${FILE_NAME}.log"
                     filePattern="${sys:user.home}/logs/$${date:yyyy-MM}/${FILE_NAME}-%d{yyyy-MM-dd}-%i.log">
            <PatternLayout pattern="${CONSOLE_PATTERN}"/>
            <Policies>
                <TimeBasedTriggeringPolicy/>
                <SizeBasedTriggeringPolicy size="100 MB"/>
            </Policies>
        </RollingFile>
    </appenders>
    <loggers>
        <!--过滤掉spring和mybatis的一些无用的DEBUG信息-->
        <logger name="org.springframework" level="INFO"></logger>
        <logger name="org.mybatis" level="INFO"></logger>
        <logger name="org.apache.http" level="INFO"></logger>
        <logger name="org.apache.kafka" level="INFO"></logger>
        <logger name="com.netflix.discovery" level="INFO"></logger>
        <logger name="org.hibernate" level="INFO"></logger>
        <root level="${log.level}">
            <appender-ref ref="Console"/>
            <appender-ref ref="RollingFileInfo"/>
        </root>
    </loggers>
</configuration>
```

mysql初始化扫描配置类com.heima.article.config.MysqlConfig

```
@Configuration
@ComponentScan("com.heima.common.mysql.core")
public class MysqlConfig {

}
```

7.3.3 article服务功能开发

（0）定义dto

```
@Data
```

```
    // 省市
    Integer provinceId;
    // 市区
    Integer cityId;
    // 区县
    Integer countyId;
    // 最大时间
    Date maxBehotTime;
    // 最小时间
    Date minBehotTime;
    // 分页size
    Integer size;
    // 数据范围，比如频道ID
    String tag;

}
```

（1）接口定义,在apis模块中我们建立包**com.heima.article.apis**并定义接口ArticleHomeControllerApi

```
/**
 * 首頁文章
 */
public interface ArticleHomeControllerApi {
    /**
     * 加載首頁文章
     * @param dto 封装参数对象
     * @return 文章列表数据
     */
    ResponseResult load(ArticleHomeDto dto);

    /**
     * 加载更多
     * @param dto 封装参数对象
     * @return 文章列表数据
     */
    ResponseResult loadMore(ArticleHomeDto dto);

    /**
     * 加载最新的数据
     * @param dto 封装参数对象
     * @return 文章列表
     */
    ResponseResult loadNew(ArticleHomeDto dto);

}
```

（2）在定义完数据接口之后，我们需要做的就是去article模块定义我们的控制器， 如果你是拷贝的下面代码你可能发现你的代码中ArticleIndexService没有定义，报错了这里不用慌我们后面就是service层的编写，后面service到dao层也是同样的

```
@RestController
@RequestMapping("/api/v1/article")
public class ArticleHomeController implements ArticleHomeControllerApi {
```

```
    private AppArticleService appArticleService;



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

    @Override
    @GetMapping("/loadmore")
    public ResponseResult loadMore(ArticleHomeDto dto) {
        return appArticleService.load( ArticleConstans.LOADTYPE_LOAD_MORE, dto);
    }

    @Override
    @GetMapping("/loadnew")
    public ResponseResult loadNew(ArticleHomeDto dto) {
        return appArticleService.load( ArticleConstans.LOADTYPE_LOAD_NEW, dto);
    }

}
```

定义常量com.heima.common.article.constans.ArticleConstans

```
public class ArticleConstans{
    public static final Short LOADTYPE_LOAD_MORE = 1;
    public static final Short LOADTYPE_LOAD_NEW = 2;
    public static final String DEFAULT_TAG = "__all__"

}
```

（3）文章service接口定义

```
public interface AppArticleService {

    /**
     *
     * @param type 1 加载更多  2 加载更新
     * @param dto 封装数据
     * @return 数据列表
     */
    public ResponseResult load(Short type, ArticleHomeDto dto);
}
```

（4）AppArticleServiceImpl实现类

```
@Service
public class AppArticleServiceImpl implements AppArticleService {

    // 单页最大加载的数字
```

```
    @Autowired
    private ApArticleMapper apArticleMapper;
    @Autowired
    private ApUserArticleListMapper apUserArticleListMapper;


    /**
     *
     * @param time 时间节点
     * @param type 1 加载更多  2 加载更新
     * @param size 每次返回数据量
     * @return 数据列表
     */
    public ResponseResult load(Short type, ArticleHomeDto dto) {
        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());
        }
        // 数据加载
        if(user!=null){
            return ResponseResult.okResult(getUserArticle(user,dto,type));
        }else{
            return ResponseResult.okResult(getDefaultArticle(dto,type));
        }
    }

    /**
     * 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
     */
```

```
        List<ApUserArticleList> list =
apUserArticleListMapper.loadArticleIdListByUser(user,dto,type);
        if(!list.isEmpty()){
            List<ApArticle> temp =
apArticleMapper.loadArticleListByIdList(list);
            return temp;
        }else{
            return getDefaultArticle(dto,type);
        }
    }

    /**
     * 从默认的大文章列表中获取文章
     * @param dto
     * @param type
     * @return
     */
    private List<ApArticle> getDefaultArticle(ArticleHomeDto dto,Short type){
        return apArticleMapper.loadArticleListByLocation(dto,type);
    }

}
```

(5) ApArticleMapper

```
public interface ApArticleMapper {


    /**
     * 照用户地理位置，加载文章
     * @param dto   参数封装对象
     * @param type  加载方向
     * @return
     */
    List<ApArticle> loadArticleListByLocation(@Param("dto") ArticleHomeDto dto,
@Param("type") short type);

    /**
     * 依据文章IDS来获取文章详细内容
     * @param list   文章ID
     * @return
     */
    List<ApArticle> loadArticleListByIdList(@Param("list")
List<ApUserArticleList> list);


}
```

(6) ApArticleMapper 对应xml

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

```
        <result column="title" property="title"/>
        <result column="author_id" property="authorId"/>
        <result column="author_name" property="authorName"/>
        <result column="channel_id" property="channelId"/>
        <result column="channel_name" property="channelName"/>
        <result column="layout" property="layout"/>
        <result column="flag" property="flag"/>
        <result column="images" property="images"/>
        <result column="labels" property="labels"/>
        <result column="likes" property="likes"/>
        <result column="collection" property="collection"/>
        <result column="comment" property="comment"/>
        <result column="views" property="views"/>
        <result column="province_id" property="provinceId"/>
        <result column="city_id" property="cityId"/>
        <result column="county_id" property="countyId"/>
        <result column="created_time" property="createdTime"/>
        <result column="publish_time" property="publishTime"/>
        <result column="sync_status" property="syncStatus"/>
    </resultMap>
    <sql id="Base_Column_List">
    id, title, author_id, author_name, channel_id, channel_name, layout, flag,
images,
    labels, likes, collection, comment, views, province_id, city_id, county_id,
created_time,
    publish_time,sync_status
  </sql>

    <!-- 依据地理位置获取 -->
    <select id="loadArticleListByLocation" resultMap="resultMap">
        select * from ap_article a
        <where>
            <if test="dto.provinceId!=null">
                and a.province_id=#{dto.provinceId}
            </if>
            <if test="dto.cityId!=null">
                and a.city_id=#{dto.cityId}
            </if>
            <if test="dto.countyId!=null">
                and a.county_id=#{dto.countyId}
            </if>
            <!-- loadmore -->
            <if test="type != null and type == 1">
                and a.publish_time <![CDATA[<]]> #{dto.minBehotTime}
            </if>
            <if test="type != null and type == 2">
                and a.publish_time <![CDATA[>]]> #{dto.maxBehotTime}
            </if>
            <if test="dto.tag != '__all__'">
                and a.channel_id = #{dto.tag}
            </if>
        </where>
        limit #{dto.size}
    </select>

    <!-- 以及文章IDS列表获取文章数据 -->
```

```
        <trim prefix="" suffixOverrides=",">
            <foreach item="item" collection="list" separator=",">
                #{item.articleId},
            </foreach>
        </trim>
        )
    </select>
</mapper>
```

（7）ApUserArticleListMapper

```
package com.heima.article.mysql.core.model.mappers.app;

import com.heima.article.mysql.core.model.dtos.ArticleHomeDto;
import com.heima.article.mysql.core.model.pojos.app.ApUser;
import com.heima.article.mysql.core.model.pojos.app.ApUserArticleList;
import org.apache.ibatis.annotations.Param;

import java.util.List;

public interface ApUserArticleListMapper {
    /**
     * 按照用户属性阅读习惯，加载文章id
     * @param user  当前登录的用户
     * @param dto   参数封装对象
     * @param type  加载方向
     * @return
     */
    List<ApUserArticleList> loadArticleIdListByUser(@Param("user") ApUser user,
@Param("dto") ArticleHomeDto dto, @Param("type") short type);

}
```

（8）ApUserArticleListMapper 对应xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper
namespace="com.heima.article.mysql.core.model.mappers.app.ApUserArticleListMappe
r">
<resultMap id="BaseResultMap"
type="com.heima.model.user.pojos.ApUserArticleList">
        <id column="id" property="id"/>
        <result column="user_id" property="userId"/>
        <result column="channel_id" property="channelId"/>
        <result column="article_id" property="articleId"/>
        <result column="is_show" property="isShow" javaType="java.lang.Boolean"
jdbcType="BIT" />
        <result column="recommend_time" property="recommendTime"
javaType="java.util.Date" jdbcType="TIMESTAMP" />
        <result column="is_read" property="isRead" javaType="java.lang.Boolean"
jdbcType="BIT" />
        <result column="strategy_id" property="strategyId"/>
```

```
    id, user_id, channel_id, article_id, is_show, recommend_time, is_read,
strategy_id
  </sql>
  <select id="loadArticleIdListByUser" parameterType="map"
resultMap="BaseResultMap">
    select
    <include refid="Base_Column_List" />
    from ap_user_article_list
    <where>
      user_id=#{user.id} and is_show=0 and is_read=0
      <!-- loadmore -->
      <if test="type != null and type == 1">
          and recommend_time <![CDATA[<]]> #{dto.minBehotTime}
      </if>
      <if test="type != null and type == 2">
          and recommend_time <![CDATA[>]]> #{dto.maxBehotTime}
      </if>
      <if test="dto.tag != '__all__'">
          and channel_id = #{dto.tag}
      </if>
  </where>
    limit #{dto.size}
  </select>
</mapper>
```

（9）单元测试

```
/**
 * 测试文章列表相关接口
 */
@SpringBootTest(classes = ArticleJarApplication.class)
@RunWith(SpringJUnit4ClassRunner.class)
public class ArticleTest {

    @Autowired
    private AppArticleService appArticleService;

    /**
     * 测试load
     */
    @Test
    public void testLoad() {
        ApUser apUser = new ApUser();
        apUser.setId(11);
        AppThreadLocalUtils.setUser(apUser);
        ArticleHomeDto dto = new ArticleHomeDto();
        ResponseResult data = appArticleService.load(
ArticleConstans.LOADTYPE_LOAD_MORE, dto);
        System.out.println(data.getData());
    }

}
```

7.3.4 行为相关功能开发

导入heima-leadnews-behavior工程

dto的定义：

com.heima.model.behavior.dtos.ShowBehaviorDto

```
@Data
public class ShowBehaviorDto {
    // 设备ID
    @IdEncrypt
    Integer equipmentId;
    List<ApArticle> articleIds;
}
```

定义控制器以及控制器接口

（1）接口定义，com.heima.article.apis.BehaviorControllerApi

```
/**
 * 行为
 */
public interface BehaviorControllerApi {

    ResponseResult saveShowBehavior(ShowBehaviorDto dto);

}
```

（2）定义控制器：com.heima.behavior.controller.v1.BehaviorController

```
@RestController
@RequestMapping("/api/v1/behavior")
public class BehaviorController implements BehaviorControllerApi {
    @Autowired
    private AppShowBehaviorService appShowBehaviorService;

    @Override
    @PostMapping("/show_behavior")
    public ResponseResult saveShowBehavior(@RequestBody ShowBehaviorDto dto) {
        return appShowBehaviorService.saveShowBehavior(dto);
    }

}
```

（3）行为服务层接口：com.heima.behavior.service.AppShowBehaviorService

```
public interface AppShowBehaviorService {

    /**
     * 存储行为数据
     * @param dto
     * @return
     */
    public ResponseResult saveShowBehavior(ShowBehaviorDto dto);

}
```

（4）AppShowBehaviorService 实现

```
@Service
@SuppressWarnings("all")
public class AppShowBehaviorServiceImpl implements AppShowBehaviorService {

    @Autowired
    private ApShowBehaviorMapper apShowBehaviorMapper;
    @Autowired
    private ApBehaviorEntryMapper apBehaviorEntryMapper;

    @Override
    public ResponseResult saveShowBehavior(ShowBehaviorDto dto){
        ApUser user = AppThreadLocalUtils.getUser();
        // 用户和设备不能同时为空
        if(user==null&&
(dto.getArticleIds()==null||dto.getArticleIds().isEmpty())){
            return ResponseResult.errorResult(AppHttpCodeEnum.PARAM_REQUIRE);
        }
        Long userId = null;
        if(user!=null){
            userId = user.getId();
        }
        ApBehaviorEntry apBehaviorEntry =
apBehaviorEntryMapper.selectByUserIdOrEquipment(userId, dto.getEquipmentId());
        // 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 ResponseResult.errorResult(AppHttpCodeEnum.PARAM_INVALID);
        }
        // 过滤新数据
        Integer[] temp = new Integer[dto.getArticleIds().size()];
        for (int i = 0; i < temp.length; i++) {
            temp[i]=dto.getArticleIds().get(i).getId();
        }
        List<ApShowBehavior> list =
apShowBehaviorMapper.selectListByEntryIdAndArticleIds(apBehaviorEntry.getId(),
temp);
        List<Integer> stringList = new ArrayList(Arrays.asList(temp));
        if(!list.isEmpty()){
            list.forEach(item->{
                stringList.remove(item.getArticleId());
            });
        }
        // 插入新数据
```

```
            stringList.toArray(temp);
            apShowBehaviorMapper.saveBehaviors(apBehaviorEntry.getId(), temp);
        }
        return ResponseResult.okResult(0);
    }
}
```

（5）heima-leadnews-model中定义行为mapper接口：
com.heima.article.mysql.core.model.mappers.app.ApShowBehaviorMapper

```
public interface ApShowBehaviorMapper {
    /**
     * 获取以及存在的用户数据
     * @param entryId
     * @param articleIds
     * @return
     */
    List<ApShowBehavior> selectListByEntryIdAndArticleIds(@Param("entryId")
Integer entryId, @Param("articleIds") Integer[] articleIds);

    /**
     * 保存用户展现行为数据
     * @param articleIds  文章IDS
     * @param entryId 实体ID
     */
    void saveBehaviors(@Param("entryId") Integer entryId, @Param("articleIds")
Integer[] articleIds);
}
```

（6）heima-leadnews-model中定义行为mapper文件：
mappers/app/ApShowBehaviorMapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper
namespace="com.heima.article.mysql.core.model.mappers.app.ApShowBehaviorMapper"
>
  <resultMap id="BaseResultMap"
type="com.heima.article.mysql.core.model.pojos.app.ApShowBehavior" >
      <id column="id" property="id" />
      <result column="entry_id" property="entryId" />
      <result column="article_id" property="articleId" />
      <result column="is_click" property="isClick"/>
      <result column="show_time" property="showTime" />
      <result column="created_time" property="createdTime" />
  </resultMap>
  <sql id="Base_Column_List" >
    id, entry_id, article, is_click, show_time, created_time
  </sql>


  <!-- 选择用户的行为对象，优先按用户选择 -->
  <select id="selectListByEntryIdAndArticleIds" resultMap="BaseResultMap" >
```

```
    <foreach item="item" collection="articleIds" separator=",">
      #{item}
    </foreach>
    )
  </select>

  <insert id="saveBehaviors">
  /*!mycat:catlet=io.mycat.route.sequence.BatchInsertSequence */
  insert into ap_show_behavior ( entry_id, article_id,is_click, show_time,
created_time) values
    <foreach item="item" collection="articleIds" separator=",">
      (#{entryId}, #{item},0, now(),now())
    </foreach>
  </insert>
</mapper>
```

(7) ApBehaviorEntryMapper

```
public interface ApBehaviorEntryMapper {

    ApBehaviorEntry selectByUserIdOrEquipment(@Param("userId") Integer userId,
@Param("equipmentId") Integer equipmentId);

}
```

(8) ApBehaviorEntryMapper 对应映射配置

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper
namespace="com.heima.article.mysql.core.model.mappers.app.ApBehaviorEntryMapper"
>
  <resultMap id="BaseResultMap"
type="com.heima.article.mysql.core.model.pojos.app.ApBehaviorEntry" >
      <id column="id" property="id" />
      <result column="type" property="type"/>
      <result column="entry_id" property="entryId" />
      <result column="created_time" property="createdTime" />
      <result column="burst" property="burst"/>
  </resultMap>
  <sql id="Base_Column_List" >
    id, type, entry_id, created_time
  </sql>
  <!-- 选择用户的行为对象，优先按用户选择 -->
  <select id="selectByUserIdOrEquipment" resultMap="BaseResultMap" >
    select * from ap_behavior_entry a
    <where>
      <if test="userId!=null">
        and a.entry_id=#{userId} and type=1
      </if>
      <if test="userId==null and equipmentId!=null">
        and a.entry_id=#{equipmentId} and type=0
      </if>
```

```
  </select>
</mapper>
```

行为接口测试

```
/**
 * 测试文章列表相关接口
 */
@SpringBootTest(classes = BehaviorJarApplication.class)
@RunWith(SpringJUnit4ClassRunner.class)
public class BehaviorTest {


    @Autowired
    private AppShowBehaviorService showBehaviorService;


    @Test
    public void testSaveBehavior() {
        ApUser apUser = new ApUser();
        apUser.setId(1l);
        AppThreadLocalUtils.setUser(apUser);
        ShowBehaviorDto dto = new ShowBehaviorDto();
        List<ApArticle> articles = new ArrayList<>();
        ApArticle apArticle = new ApArticle();
        apArticle.setId(1);
        articles.add(apArticle);
        showBehaviorService.saveShowBehavior(dto);
        //articleIndexService.saveBehaviors(data);
    }
}
```