即刻 开发文档

> 即刻 (JIKE) App 是一款专注年轻人社交的App,以同好圈子的社交概念以及友好的交互设计拉进用户间的距离,鼓励用户造有价值的内容认识更多趣味相投的朋友.
>
> 本项目尝试实现 即刻App 的H5移动端版本.

需求分析

> 根据 市面上已有的 即刻 App 我们分析出以下目前能够实现的需求

登录/注册

注册: 用户通过填写相关信息进行注册操作成为正式用户

登录: 用户通过填写账号相关信息以及通过验证登录

用户的个人信息修改

用户登录后可对自己的昵称,头像,签名等信息进行修改

动态展示

展示用户发布在圈子内的动态,以及该动态获得的点赞数和评论数和评论详情

发布动态

用户可将编辑的动态发布在选定的圈子内供他人查看

点赞

用户可对其认为值得点赞的动态进行点赞操作,动态的点赞数将会被累加

评论

用户可对动态发表自己的想法和意见

表情

用户在发布动态和回复他人时可使用心仪的表情

关注与被关注

用户可关注他人也可被他人关注,关注的用户发布的动态将会出现在关注动态中

用户卡片

展示用户基本信息的卡片

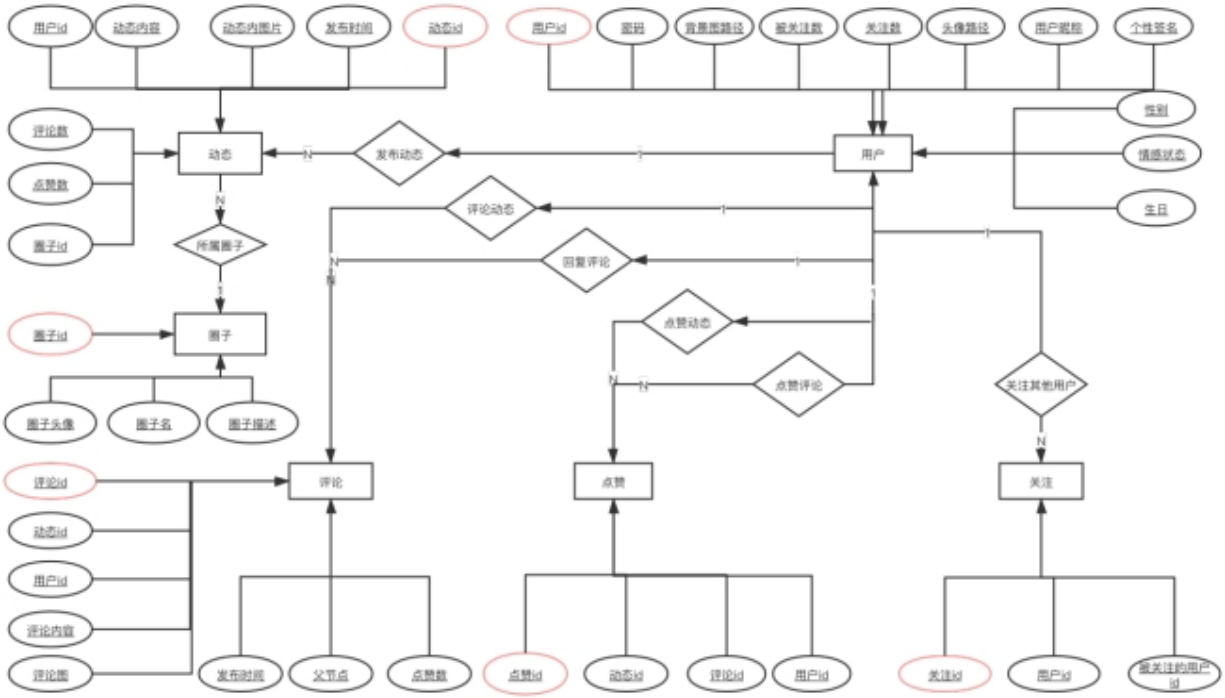
概要设计

> 根据需求分析,我们先进行数据库设计,接着再是原型图设计,最后我们将需求基本分为几大模块来进行实现.每个组员负责不同的模块,遵从文档接口进行开发

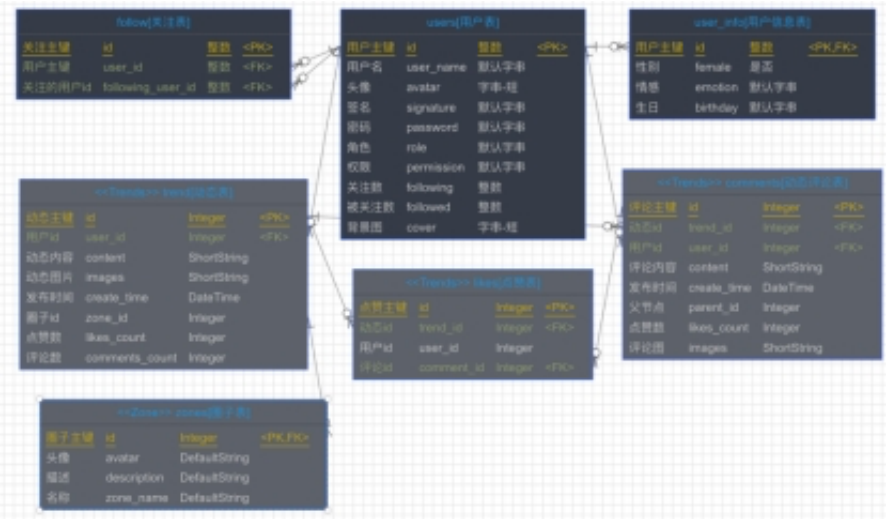
数据库设计

> 根据需求分析以及分析即刻App的数据体现, 画出数据库的ER图,根据ER图构建数据库模型

ER图如下

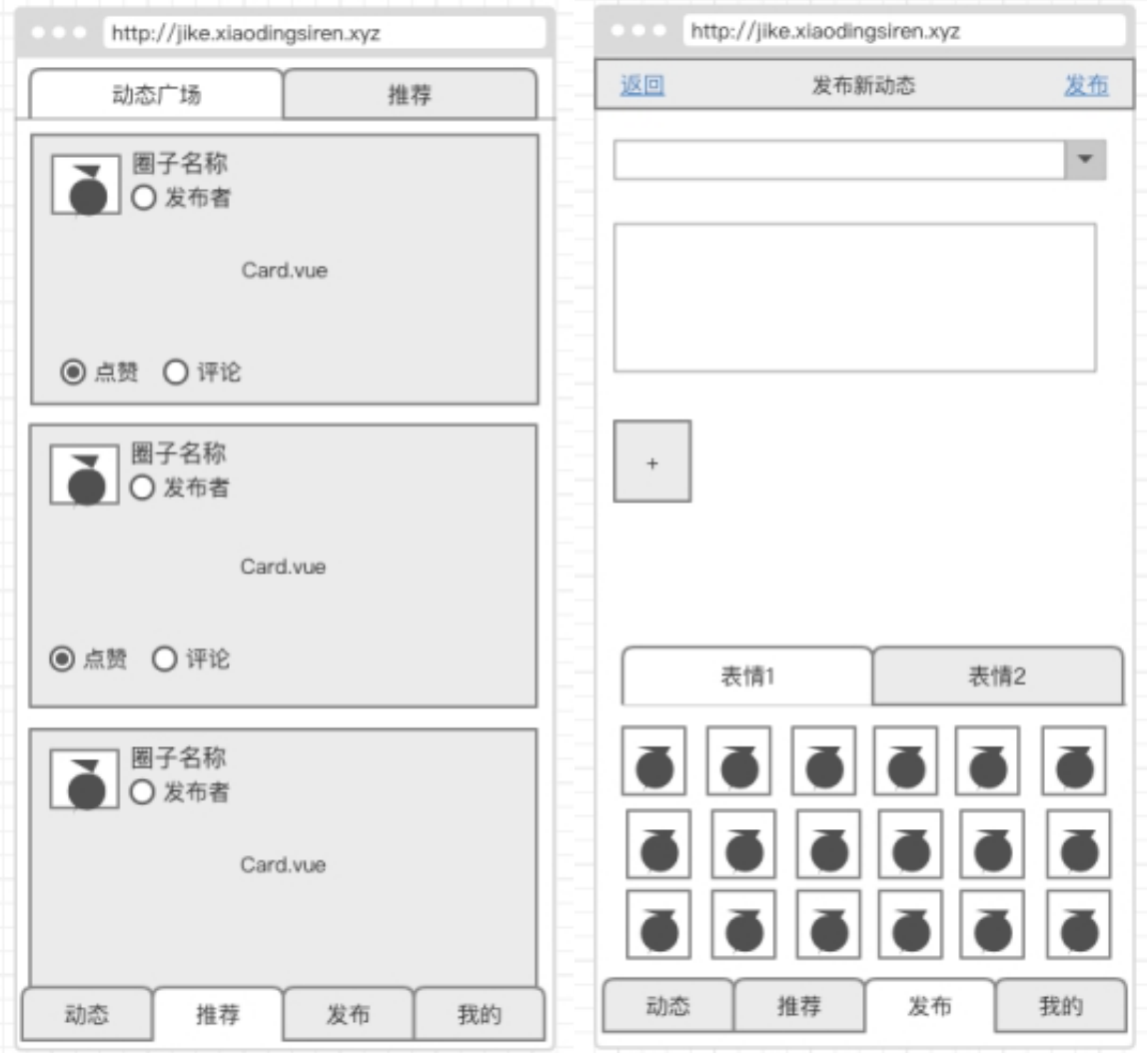


数据库模型如下



原型图设计

> 参照即刻App 的设计,我们很快得出了该项目的原型图


http://jike.xiaodingsiren.xyz
动态广场
推荐
圈子名称
发布者
Card.vue
点赞
评论
圈子名称
发布者
Card.vue
点赞
评论
圈子名称
发布者
Card.vue
动态
推荐
发布
我的
http://jike.xiaodingsiren.xyz
返回
发布新动态
发布
+
表情1
表情2
动态
推荐
发布
我的


http://jike.xiaodingsiren.xyz
用户名
0 关注| 0 被关注
动态
档案
圈子名称
发布者
Card.vue
点赞
评论
圈子名称
发布者
Card.vue
http://jike.xiaodingsiren.xyz
动态
圈子名称
发布者
Card.vue
点赞
评论
圈子名称
发布者
Card.vue
点赞
评论
圈子名称
发布者
Card.vue

模块设计

用户模块

主要完成用户的登录/注册,用户信息的获取和修改等需求

接口如下

用户模块共 (6) 个

添加接口

接口名称	接口路径	接口分类	状态	tag
注册	POST /users	用户模块	已完成	未设置
登录	POST /users/login	用户模块	已完成	未设置
获取用户卡片	GET /users/info/cards	用户模块	已完成	未设置
获取用户档案	GET /users/personal/others	用户模块	已完成	未设置
修改用户档案	POST /users/personal	用户模块	已完成	未设置
获取用户自己的档案	GET /users/personal	用户模块	已完成	未设置

< 1 >

动态模块

主要完成动态的获取,发布等需求

接口如下

动态模块共 (5) 个

添加接口

接口名称	接口路径	接口分类	状态	tag
获取其他用户发布的动态	GET /trend/others	动态模块	已完成	未设置
获取推荐流信息	GET /trend/recommend	动态模块	已完成	未设置
发布动态	POST /trend	动态模块	已完成	未设置
获取已关注用户发布的动态	GET /trend/following	动态模块	已完成	未设置
获取用户点赞过的动态	GET /like/trend	动态模块	已完成	未设置

< 1 >

点赞关注模块

主要为完成动态,评论点赞的功能以及关注其他用户的需求

点赞关注模块共 (7) 个　　添加接口

接口名称	接口路径	接口分类	状态	tag
获取其他用户关注列表	GET /follow/ing/{userId}	点赞关注模块	未完成	未设置
点赞动态	POST /like/trend	点赞关注模块	已完成	未设置
关注用户	POST /follow	点赞关注模块	已完成	未设置
取消动态点赞	DELETE /like/trend	点赞关注模块	已完成	未设置
取消评论点赞	DELETE /like/comm	点赞关注模块	未完成	未设置
点赞评论	POST /like/comm	点赞关注模块	已完成	未设置
获取已关注的用户	GET /follow	点赞关注模块	已完成	未设置

评论模块

主要完成评论动态以及回复等需求

接口如下

评论模块共 (3) 个　　添加接口

接口名称	接口路径	接口分类	状态	tag
获取普通评论	GET /comm	评论模块	已完成	未设置
发布评论	POST /comm	评论模块	已完成	未设置
获取用户点赞过的评论	GET /like/comm	评论模块	已完成	未设置

< 1 >

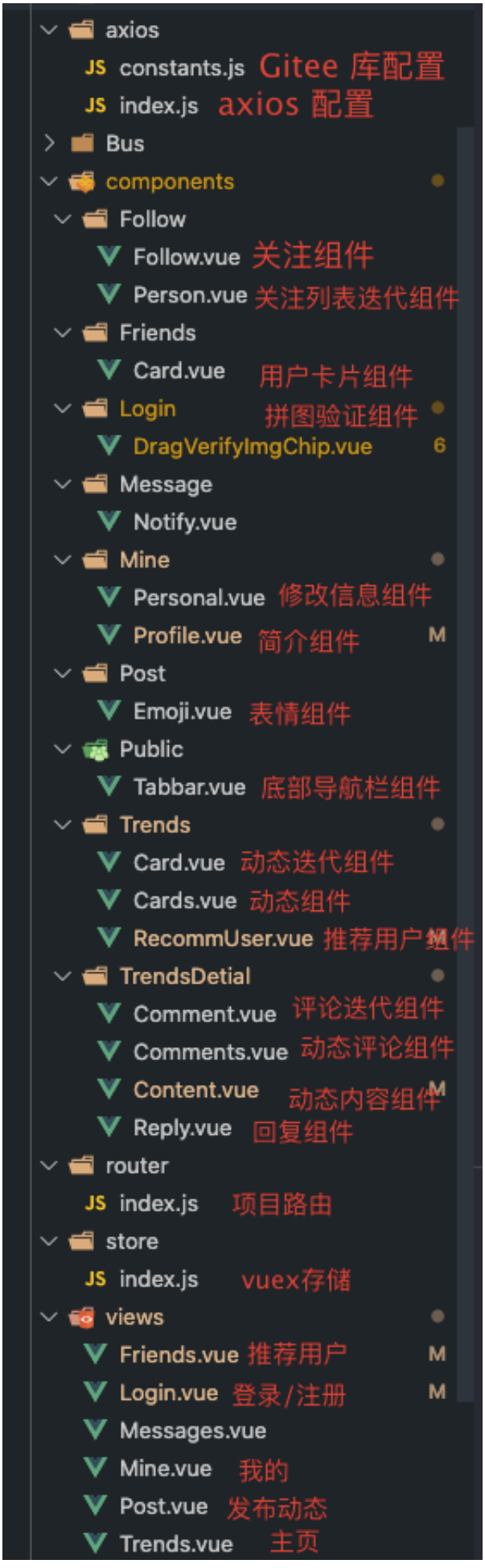
详细设计

原型实现

前端

1. 使用Vue-Cli3 创建项目

目录结构如下


axios
constants.js Gitee 库配置
index.js axios 配置
Bus
components
Follow
Follow.vue 关注组件
Person.vue 关注列表迭代组件
Friends
Card.vue 用户卡片组件
Login 拼图验证组件
DragVerifyImgChip.vue 6
Message
Notify.vue
Mine
Personal.vue 修改信息组件
Profile.vue 简介组件 M
Post
Emoji.vue 表情组件
Public
Tabbar.vue 底部导航栏组件
Trends
Card.vue 动态迭代组件
Cards.vue 动态组件
RecommUser.vue 推荐用户组件
TrendsDetial
Comment.vue 评论迭代组件
Comments.vue 动态评论组件
Content.vue 动态内容组件 M
Reply.vue 回复组件
router
index.js 项目路由
store
index.js vuex存储
views
Friends.vue 推荐用户 M
Login.vue 登录/注册 M
Messages.vue
Mine.vue 我的
Post.vue 发布动态
Trends.vue 主页

2. 配置axios,vuex,router

    1. axios 基本配置,响应403未授权操作

```
// 超时设置
axios.defaults.timeout = 15000
// 请求头设置
axios.defaults.headers.post['Content-Type'] = 'application/json;charset=UTF-8';
axios.defaults.headers.get['Content-Type'] = 'application/json;charset=UTF-8';
axios.defaults.baseURL = '/'
/* 请求之后的操作 */
axios.interceptors.response.use((res) => {
    console.log(res)
    return res;
}, (error) => {
    console.log(error)
    if (error.response.data.message == '403') {
        Notify({ type: 'warning', message: '登录已失效' });
        store.commit('setToken', null)
    }
    return Promise.reject(error);
});
```

    2. 配置 vuex

```
// 引入vuex-persistedstate 持久化库,解决vuex 刷新时效问题
import createPersistedState from 'vuex-persistedstate';
```

```
3
4   const store = new Vuex.Store({
5       state: {
6           token: null, //缓存token
7           emojiList: { acList: [], aluList: [] }, //对表情列表进行缓存
8       },
9       plugins: [createPersistedState()],
10      mutations: {
11          setToken: (state, token) => { state.token = token },
12          setAcList: (state, palyload) => { state.emojiList.acList = palyload.acList },
13          setAluList: (state, palyload) => { state.emojiList.aluList = palyload.aluList },
14      },
15      getters: {
16          acList: state => {
17              return state.emojiList.acList
18          },
19          aluList: state => {
20              return state.emojiList.aluList
21          },
22          token: state => {
23              return state.token
24          }
25      }
26  });
```

3. 配置router

```
1   const routes = [
2       {
3           path: '/',
4           name: 'Trends',
```

```
            component: () =>
    import('../views/Trends.vue'),
            meta: { title: "推荐" }
        },
        {
            path: '/Friends',
            name: 'Friends',
            component: () =>
    import('../views/Friends.vue'),
            meta: { title: "朋友" }
        },
        {
            path: '/Post',
            name: 'Post',
            component: () =>
    import('../views/Post.vue'),
            meta: { title: "发布动态" }
        },
        {
            path: '/Messages',
            name: 'Messages',
            component: () =>
    import('../views/Messages.vue'),
            meta: { title: "通知" }
        },
        {
            path: '/Mine',
            name: 'Mine',
            component: () =>
    import('../views/Mine.vue'),
            meta: { title: "我的" }
        },
        {
            path: '/Others/:userId',
            name: 'Others',
```

```
35            component: () =>
    import('../views/Mine.vue'),
36            meta: { title: "名片" }
37        },
38        {
39            path: '/TrendsDetial/:id',
40            name: 'TrendsDetial',
41            component: () =>
    import('../views/TrendsDetial.vue'),
42            meta: { title: "即刻动态" }
43        },
44        {
45            path: '/Login',
46            name: 'Login',
47            component: () =>
    import('../views/Login.vue'),
48            meta: { title: "登录/注册" }
49        },
50        {
51            path: '/Personal',
52            name: 'Personal',
53            component: () =>
    import('../components/Mine/Personal.vue'),
54            meta: { title: "编辑个人信息" }
55        },
56        {
57            path: '/Following/:userId',
58            name: 'Following',
59            component: () =>
    import('../components/Follow/Follow.vue'),
60            meta: { title: "正在关注" }
61        },
62        {
63            path: '/Followed/:userId',
64            name: 'Followed',
```

```
            component: () =>
    import('../components/Follow/Follow.vue'),
            meta: { title: "正被关注" }
        }
    ]
```

3. 引入vant UI,本地开发配置

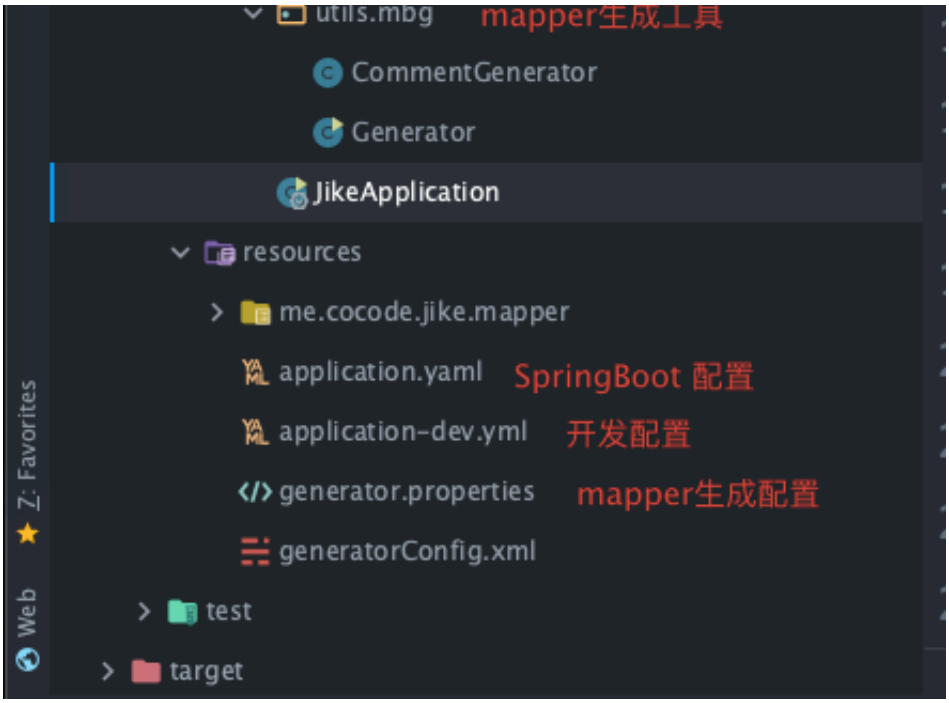
```
// main.js
import { vant } from 'vant';
Vue.use(vant);
// vue.config.js
devServer: {
        open: true,
        host: "0.0.0.0",
        proxy: {
            '/gitee': { // gitee 库
                target:
'https://gitee.com/api/v5/repos/xiaodingsiren/JikePic/contents/', // giteeApi
                changeOrigin: true,
                pathRewrite: {
                    '^/gitee': '' //
localhost:8080/gitee/xxx =>
https://gitee.com/api/v5/repos/xiaodingsiren/JikePic/contents/xxx
                }
            },
            '/jike-api': { // 项目后台
                target:
'http://127.0.0.1:8081/jike-api',
                changeOrigin: true,
                pathRewrite: {
                    '^/jike-api': ''
                }
```

```
22                  }
23              }
24          },
```

后端

1. 创建maven 项目,引入依赖,配置 application.yml

   项目 结构如下



引入依赖

```
 <dependencies>
        <!-- SpringBoot 核心 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-aop</artifactId>
```

```
15          </dependency>
16          <dependency>
17
     <groupId>org.springframework</groupId>
18              <artifactId>spring-context-
    support</artifactId>
19          </dependency>
20          <dependency>
21
     <groupId>org.springframework.boot</groupId>
22              <artifactId>spring-boot-starter-
    data-redis</artifactId>
23          </dependency>
24          <dependency>
25
     <groupId>org.springframework.boot</groupId>
26              <artifactId>spring-boot-
    configuration-processor</artifactId>
27              <optional>true</optional>
28          </dependency>
29
30          <dependency>
31
     <groupId>org.springframework.boot</groupId>
32              <artifactId>spring-boot-
    devtools</artifactId>
33              <scope>runtime</scope>
34              <optional>true</optional>
35          </dependency>
36          <dependency>
37              <groupId>org.projectlombok</groupId>
38              <artifactId>lombok</artifactId>
39              <optional>true</optional>
40          </dependency>
41          <dependency>
```

```
42
     <groupId>org.mybatis.spring.boot</groupId>
43                 <artifactId>mybatis-spring-boot-
    starter</artifactId>
44                 <version>2.1.4</version>
45             </dependency>
46             <dependency>
47
     <groupId>org.mybatis.generator</groupId>
48                 <artifactId>mybatis-generator-
    core</artifactId>
49                 <version>1.3.7</version>
50             </dependency>
51             <!--
    https://mvnrepository.com/artifact/tk.mybatis/map
    per-generator -->
52             <dependency>
53                 <groupId>tk.mybatis</groupId>
54                 <artifactId>mapper-
    generator</artifactId>
55                 <version>1.1.5</version>
56             </dependency>
57             <dependency>
58                 <groupId>tk.mybatis</groupId>
59                 <artifactId>mapper-spring-boot-
    starter</artifactId>
60                 <version>2.1.5</version>
61             </dependency>
62             <!-- mysql驱动 -->
63             <dependency>
64                 <groupId>mysql</groupId>
65                 <artifactId>mysql-connector-
    java</artifactId>
66                 <version>8.0.16</version>
67             </dependency>
```

```
68          <!-- druid -->
69          <dependency>
70              <groupId>com.alibaba</groupId>
71              <artifactId>druid-spring-boot-
    starter</artifactId>
72              <version>1.1.13</version>
73          </dependency>
74          <!-- swagger2 -->
75          <dependency>
76              <groupId>io.springfox</groupId>
77              <artifactId>springfox-boot-
    starter</artifactId>
78              <version>3.0.0</version>
79          </dependency>
80
81          <!--MyBatis分页插件starter-->
82          <dependency>
83
     <groupId>com.github.pagehelper</groupId>
84              <artifactId>pagehelper-spring-boot-
    starter</artifactId>
85              <version>1.2.10</version>
86          </dependency>
87          <!--MyBatis分页插件-->
88          <dependency>
89
     <groupId>com.github.pagehelper</groupId>
90              <artifactId>pagehelper</artifactId>
91              <version>5.1.8</version>
92          </dependency>
93          <!-- json  -->
94          <dependency>
95              <groupId>com.alibaba</groupId>
96              <artifactId>fastjson</artifactId>
97              <version>1.2.58</version>
```

```
98          </dependency>
99
100         <!--
    https://mvnrepository.com/artifact/org.apache.shi
    ro/shiro-spring -->
101         <dependency>
102             <groupId>org.apache.shiro</groupId>
103             <artifactId>shiro-
    spring</artifactId>
104             <version>1.4.2</version>
105         </dependency>
106
107         <!--
    https://mvnrepository.com/artifact/com.auth0/java
    -jwt -->
108         <dependency>
109             <groupId>com.auth0</groupId>
110             <artifactId>java-jwt</artifactId>
111             <version>3.8.3</version>
112         </dependency>
113
114         <dependency>
115
     <groupId>org.springframework.boot</groupId>
116             <artifactId>spring-boot-starter-
    amqp</artifactId>
117         </dependency>
118     </dependencies>
```

配置 application.yaml

```
// 配置数据源
spring:
  datasource:
    type: com.alibaba.druid.pool.DruidDataSource
    druid:
      driver-class-name: com.mysql.cj.jdbc.Driver
      url: jdbc:mysql://localhost:3306/jike?useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai
      username: root
      password: a1519877813
```

2. 整合 shrio

    1. 接管Filter

```
  @Bean("shiroFilter")
     public ShiroFilterFactoryBean factoryBean(DefaultWebSecurityManager securityManager){
         ShiroFilterFactoryBean factoryBean = new ShiroFilterFactoryBean();
         Map<String, Filter>     filerMap    = new HashMap<>();
         // 添加自己的过滤器并且取名为jwt
         filerMap.put("jwt",new JwtFilter());
         factoryBean.setFilters(filerMap);


          factoryBean.setSecurityManager(securityManager);
         /*
          * 自定义url规则
          * http://shiro.apache.org/web.html#urls-
          */
```

```
14          Map<String, String> filterRuleMap = new
    HashMap<>();
15          // 所有请求通过我们自己的JWT Filter
16          filterRuleMap.put("/users/login",
    "anon");
17          filterRuleMap.put("/**", "jwt");
18
     factoryBean.setFilterChainDefinitionMap(filterRul
    eMap);
19          return factoryBean;
20      }
```

2. 创建自己的BasicHttpAuthenticationFilter

   重写 `isLoginAttempt()` 根据请求头中是否含有 `Authorization` 来判断是否登录到shiro中

```
1    @Override
2        protected boolean
    isLoginAttempt(ServletRequest request,
    ServletResponse response) {
3            HttpServletRequest req          =
    (HttpServletRequest) request;
4            String             authorization =
    req.getHeader(LOGIN_SIGN);
5            return   authorization != null ;
6        }
```

   重写 `executeLogin()` 将JWT 生成的token 作为授权进行登录

```
 @Override
     protected boolean executeLogin(ServletRequest request, ServletResponse response) throws Exception {
         HttpServletRequest req    = (HttpServletRequest) request;
         String              authorization = req.getHeader(LOGIN_SIGN);

         JwtToken token = new JwtToken(authorization);
         logger.info("token ==> "+token);
         // 提交给realm进行登入，如果错误他会抛出异常并被捕获
         getSubject(request, response).login(token);
         // 如果没有抛出异常则代表登入成功，返回true
         return true;
     }
```

3. 生成通用mapper ,创建通用接口
    1. 配置MBG (Mybatis Generator)

```
1    <!--  实体类生成-->
2            <javaModelGenerator
   targetPackage="me.cocode.jike.entity"
   targetProject="src/main/java"/>
3            <!-- mapper xml 生成 -->
4            <sqlMapGenerator
   targetPackage="me.cocode.jike.mapper"
   targetProject="src/main/resources"/>
5            <!-- dao 生成  -->
6            <javaClientGenerator type="XMLMAPPER"
   targetPackage="me.cocode.jike.dao"
7
   targetProject="src/main/java"/>
8
9            <!--生成全部表tableName设为%-->
10           <table tableName="%">
11               <generatedKey column="id"
   sqlStatement="MySql" identity="true"/>
12           </table>
```

模块实现

用户模块 rao

登录注册功能

主要由UsersController类,UserService 接口定义方法实现,UsersController主要负责响应请求,UserService 处理具体业务逻辑.在UsersController 中,saveUser()响应注册请求,根据请求参数值 username 和 password 先查询是否已存在该用户调用 UserService 中 selectOneByName() 方法,如果已存在该用户则返回通用结果 fail, 如果未找到则将用户名作为盐值和密码MD5加密3次后的结果作为密码调用UserService 的 insertSelective() 方法

插入到数据库中,该方法遇到null值时会使用数据库中的默认值;核心代码如下

```
@PostMapping
@ApiOperation("用户注册")
public R saveUser(@RequestParam("userName") String userName,
                  @RequestParam("password") String password) {
  Users byName = userService.selectOneByName(userName);
  if (byName != null) {
    return R.failed("用户名已存在");
  }
  Users newUser = new Users();
  newUser.setUserName(userName);
  newUser.setPassword(new Md5Hash(password, userName, 3).toString());
  userService.insertSelective(newUser);
  userInfoMapper.insert(new UserInfo());
  return R.success(null);
}
```

用户登录功能,在UsersController 中的login() 方法中实现,首先根据请求参数值的 username 查询是否存在该用户,如不存在则直接返回通用结果fail,如果用户存在则将用户名作为盐值和密码MD5加密3次后与获取到的用户的密码比较,如果相同则返回账号错误信息,如果相同则使用JWTUtils签发一个包含用户主键id 的加密token 返回给前台.核心代码如下:

```
@PostMapping("/login")
@ApiOperation("用户登录")
public R login(@RequestParam("userName") String userName, @RequestParam("password") String password) {
  Users byName = userService.selectOneByName(userName);
```

```
    if (byName == null) {
      return R.failed("账号不存在!");
    }
    logger.info("byName => " + byName.toString());
    String crypto = new Md5Hash(password, userName,
3).toString();
    logger.info("crypto => " + crypto);
    if (byName.getPassword().equals(crypto)) {
      String token = JwtUtils.sign(byName.getId(),
crypto);
      return R.success(token, "登录成功");
    }
    return R.failed("账号或密码错误");
  }
```

获取用户的卡片

主要由UsersController 类以及 UserInfoMapper 接口实现,UserInfoMapper 接口中定义 getUsersInfoCard() 该方法返回 UserPersonalDto 集合 通过使用@Select 注解的方式书写sql语句不需要额外书写 xml 文件,简化了开发.UsersController 响应接口后.直接返回调用UserInfoMapper 的 getUserInfoCard() 结果.核心代码如下:

```
/**
* 获取推荐用户时的卡片信息
*/
@Select("SELECT u.id,u.user_name userName,u.avatar
userAvatar,u.signature,ui.gender,ui.emotion,ui.birthday
FROM users u,user_info ui WHERE u.id=ui.id")
List<UserPersonalDto> getUserInfoCard();
```

```
@GetMapping("/info/cards")
@ApiOperation("获取用户卡片")
public R<List<UserPersonalDto>> getUserCards() {
  return R.success(userInfoMapper.getUserInfoCard());
}
```

获取用户档案

主要由 UsersController 类以及 UserInfoMapper 接口实现,UserInfoMapper 中getUserPersonalInfo ()方法中使用@Param 注解指定@Select 注解中的#{userId} 参数.使用@ResultType 绑定结果集UsersController 响应请求后根据请求参数调用UserInfoMapper 中的getUserPersonalInfo()方法.核心代码如下:

```
/**
* 获取用户的档案信息
*/
@Select("SELECT u.user_name userName,u.avatar userAvatar,u.signature signature,u.cover cover,ui.gender gender,ui.emotion emotion,ui.birthday birthday FROM users u INNER JOIN user_info ui ON u.id=#{userId} and ui.id = #{userId} ")
@ResultType(UserPersonalDto.class)
UserPersonalDto getUserPersonalInfo(@Param("userId") Integer userId);
```

```
@GetMapping("/personal/others")
@ApiOperation("获取其他用户的档案")
public R<UserPersonalDto> getUserPersonal(@RequestParam("userId") Integer userId) {
  return R.success(userInfoMapper.getUserPersonalInfo(userId));
}
```

修改个人信息

主要由 UsersController 类和 UserInfoMapper 类实现;UserInfoMapper中updateUserPersonal() 方法使用@Param 注解指定 @Update 注解中的#{}等命名参数.UsersController 中 updatePersonal() 方法使用了@RequiresAuthentication 注解,要求Post请求头中必须携带Authentication 字段,如请求中无此字段将被shiro 拦截直接返回403 未授权操作,shiro 首先会根据Authentication 字段值 调用MyRealm 中的doGetAuthenticationInfo 方法进行认证操作,如果认证失败也将抛出403未授权操作.若通过了认证则使用shiro的SecurityUtils 中的getSubject方法获取当前的主体用户,再通过JwtUtils 在主体用户中的凭证获取用户主键,根据用户主键调用UserInfoMapper 中的updateUserPersonal()方法,参数则为POST 请求中的JSON 参数.核心代码如下:

```
@Update(
  "UPDATE users " +
  "INNER JOIN user_info " +
  "ON users.id = user_info.id " +
  "SET user_name=#{userName},avatar=#{userAvatar},signature=#{signature},cover=#{cover},birthday=#{birthday},gender=#{gender},emotion=#{emotion} " +
  "WHERE users.id=#{userId}")
int updateUserPersonal(@Param("userName") String userName,
                       @Param("userAvatar") String userAvatar,
                       @Param("signature") String signature,
                       @Param("cover") String cover,
                       @Param("birthday") String birthday,
                       @Param("gender") String gender,
```

```
                            @Param("emotion") String
    emotion,
                            @Param("userId") Integer
    userId);
```

```
@PostMapping("/personal")
@RequiresAuthentication
@ApiOperation("修改个人档案")
public R updatePersonal(@RequestBody UserPersonalDto
    userPersonalDto) {
  Subject subject = SecurityUtils.getSubject();
  Integer userId =
    JwtUtils.getUserId(subject.getPrincipals().toString());
  return
    R.success(userInfoMapper.updateUserPersonal(userPersona
    lDto.getUserName(),
                userPersonalDto.getUserAvatar(),
                userPersonalDto.getSignature(),
                userPersonalDto.getCover(),
                userPersonalDto.getBirthday(),
                userPersonalDto.getGender(),
                userPersonalDto.getEmotion(),
                userId));
}
```

注销

主要由UsersController 类实现,UsersController 类中 logout()方法响应请求后调用SecurityUtils中getSubject() 方法获得当前主体,如果当前主体已认证过则可进行logout 操作,否则返回注销失败信息.核心代码如下:

```
@GetMapping("/logout")
@ApiOperation("用户登出")
@RequiresAuthentication
public R logout() {
  Subject subject = SecurityUtils.getSubject();
  if (subject.isAuthenticated()) {
    subject.logout();
    return R.success(null);
  } else {
    return R.failed("登出失败,未认证!");
  }
}
```

未登录拦截

在未登录情况下,用户将无法进行发布动态,评论点赞等等需要权限的操作,为防止这种情况发生主要通过两方面来处理,前端发送需要权限的请求前先检查是否缓存了token,无则提醒用户需要登录再操作,后端则通过检查请求头Authentication 值来判断是否为合法token;核心代码如下:

```
@Override
protected AuthenticationInfo
doGetAuthenticationInfo(AuthenticationToken auth)
throws AuthenticationException {
  logger.info("认证: doGetAuthenticationInfo==>" +
auth);

  String  token  = (String) auth.getCredentials();
  if (token == null) {
    throw new
AuthenticationException(ResultCode.UNAUTHORIZED.getMess
age());
  }
  Integer userId = JwtUtils.getUserId(token);
```

```
10    if (userId == null) {
11      throw new
    AuthenticationException(ResultCode.UNAUTHORIZED.getMess
    age());
12    }
13    Users user = service.selectByPrimaryKey(userId);
14    if (user == null) {
15      throw new AuthenticationException("账号不存在!");
16    }
17    if (!JwtUtils.verify(token, userId,
    user.getPassword())) {
18      throw new AuthenticationException("账号或密码不正
    确!");
19    }
20    return new SimpleAuthenticationInfo(token, token,
    "my_realm");
21  }
```

动态模块 zeng

发布动态

主要由 TrendController 类 和 TrendService 接口实现,TrendService 中 postTrend() 方法由实现类 TrendServiceImpl 实现,先根据 PostTrendDto 传输对象获得 zoneName 圈子名字段,再构建通用查询的用例,设置选择字段为id,设置查询条件为'zoneName'字段等于 zoneName,接着根据 ZonesMapper中的selectOneByExample() 进行条件查询.得到圈子信息后将 dto 信息装载到新动态中并设置所属的圈子id;TrendController 类中 postTrend()方法要求请求头中须携带Authorization值.根据POST的参数将 JSON 字符串转换为postTrendDto 对象接着使用JWTUilts获取token中的用户id,然后根据用户id设置新动态的发布者id,最后调用 trendService 的 insertSelective()方法,属性为null的值将使用数据库默认值.核心代码如下:

```java
@Override
public Trend postTrend(PostTrendDto dto) {
  String zoneName = dto.getZone();
  Example example  = new Example(Zones.class);
  example.selectProperties("id");
  Example.Criteria criteria = example.createCriteria();
  criteria.andEqualTo("zoneName",zoneName);
  Zones zone = zonesMapper.selectOneByExample(example);
  Trend result = new Trend();
  result.setContent(dto.getContent());
  result.setImages(Arrays.toString(dto.getImages()));
  result.setCreateTime(new
Date(System.currentTimeMillis()));
  result.setZoneId(zone.getId());
  return result;
}
```

```java
@PostMapping
@RequiresAuthentication
@ApiOperation("发布动态")
public ResultCode postTrend(@RequestBody String
trendJson,@RequestHeader("Authorization") String token)
{
  PostTrendDto postTrendDto =
JSON.parseObject(trendJson, PostTrendDto.class);
  Integer userId = JwtUtils.getUserId(token);
  logger.info("Dto => "+ postTrendDto.toString());
  Trend trend = trendService.postTrend(postTrendDto);
  trend.setUserId(userId);
  logger.info("trend => " + trend.toString());
  trendService.insertSelective(trend);
  return ResultCode.SUCCESS;
}
```

获取推荐动态

主要由TrendController 类和TrendService 接口实现.TrendService 接口中的 getRecommendTrends 方法由自动注入的TrendMapper中的 getRecommendTrends 实现,而getRecommendTrends()方法使用@Select书写sql语句,首先内联动态表和用户表条件为动态表中的user_id字段为用户表中的id字段接着再内联圈子表条件为动态表中总的zone_id字段为圈子表中的id字段,最后根据发布时间降序排列;TrendController 中则直接调用trendService 的getRecommendTrends()方法返回TrendDto的集合;核心代码如下:

```
/**
* 获取推荐动态
*/
@Select("SELECT t.id trendId,z.zone_name
zoneName,z.avatar zoneAvatar,u.user_name userName,u.id
userId,u.avatar
userAvatar,t.images,t.content,t.likes_count
likesCount,t.comments_count
commentsCount,z.description,u.signature,t.create_time
createTime FROM trend t INNER JOIN users u INNER JOIN
zones z ON t.user_id=u.id AND t.zone_id=z.id ORDER BY
create_time desc")
@ResultType(TrendDto.class)
List<TrendDto> getRecommendTrends();
```

```
@GetMapping("/recommend")
@ApiOperation("获取推荐动态")
public R<List<TrendDto>> getRecommend(){
  return R.success(trendService.getRecommendTrends());
}
```

获取他人的动态

主要由TrendController 类和TrendMapper 接口实现.TrendMapper 中getTrendByUserId() 使用@Select 注解 指定sql语句在推荐动态的sql语句基础上增加用户表id 的条件限制, @Param指定命名参数 @ResultType 指定返回结果;TrendController 中 getUserPersonalTrend() 则根据请求参数中的userId 调用TrendMapper 的getTrendByUserId 方法,返回一个TrendDto集合; 核心代码如下:

```
/**
* 根据用户id获取其所发布的动态
*/
@Select("SELECT t.id trendId,z.zone_name zoneName,z.avatar zoneAvatar,u.user_name userName,u.id userId,u.avatar userAvatar,t.images,t.content,t.likes_count likesCount,t.comments_count commentsCount,t.create_time createTime FROM trend t INNER JOIN users u INNER JOIN zones z ON t.user_id=u.id AND t.zone_id=z.id WHERE u.id=#{userId} ORDER BY create_time DESC")
@ResultType(TrendDto.class)
List<TrendDto> getTrendByUserId(@Param("userId")Integer userId);
```

```
@GetMapping("/others")
@ApiOperation("获取其他用户的动态")
public R<List<TrendDto>>
 getUserPersonalTrend(@RequestParam("userId")Integer userId){
   return R.success(trendMapper.getTrendByUserId(userId));
}
```

获取已关注用户的动态

主要由TrendController 类和TrendMapper 接口实现.TrendMapper 接口中getFollowingUserTrends() 方法使用@Select 注解 sql语句首先将动态表用户表圈子表按照动态表中user_id字段等于用户表中id字段以及动态表中zone_id 字段等于圈子表中的id字段的规则内联在一起,接着结果中的id字段需在子查询表 查询用户所关注id的集合中,由此完成了对关注用户的动态查询.TrendController 类中的getFollowingUserTrends() 方法则根据请求头中的Authentication值获取到用户id,再根据用户id 去调用TrendMapper 的getFollowingUserTrends() 返回一个 TrendDto 集合; 核心代码如下:

```
/**
     * 获取已关注用户发布的动态
     */
@Select("SELECT t.id trendId,z.zone_name zoneName,z.avatar zoneAvatar,u.user_name userName,u.id userId,u.avatar userAvatar,t.images,t.content,t.likes_count likesCount,t.comments_count commentsCount,t.create_time createTime FROM trend t INNER JOIN users u INNER JOIN zones z ON t.user_id=u.id AND t.zone_id=z.id WHERE u.id in ( " +
        "SELECT following_user_id FROM follow WHERE follow.user_id=#{userId}) ORDER BY create_time DESC")
List<TrendDto> getFollowingUserTrends(@Param("userId")Integer userId);
```

```
@GetMapping("/following")
@RequiresAuthentication
@ApiOperation("获取已关注用户的动态")
public R<List<TrendDto>>  getFollowingUserTrends(){
    Subject subject = SecurityUtils.getSubject();
    Integer userId =
JwtUtils.getUserId(subject.getPrincipals().toString());
    return
R.success(trendMapper.getFollowingUserTrends(userId));
}
```

获取已点赞过的动态

主要由LikeController类和LikesMapper接口实现LikeController 类的 getLikedTrend() 方法中先根据

请求中的Authentication 值获取用户id,再构建Likes类的通用查询用例,设置查询列为trendId,设置查询条件为'userId' 等于从token中获取的用户id.最后通过LikesMapper 通用条件查询selectByExample 方法返回 一个 Likes 集合;核心代码如下:

```
@GetMapping("/trend")
@RequiresAuthentication
@ApiOperation("获取用户点赞过的动态")
public R<List<Likes>> getLikedTrend() {
    // 获取用户id
    Subject subject = SecurityUtils.getSubject();
    Integer userId =
JwtUtils.getUserId(subject.getPrincipals().toString());
    Example example = new Example(Likes.class);
    example.selectProperties("trendId");
    Example.Criteria criteria =
example.createCriteria();
    criteria.andEqualTo("userId", userId);
    return
R.success(likesMapper.selectByExample(example));
}
```

删除动态

主要由TrendController类和TrendMapper接口实现,TrendMapper 接口中deletePostedTrend 方法deletePostedTrendLikes 方法deletePostedTrendComm 方法 分别为删除动态表,点赞表,评论表中有管trendId 的字段的记录;TrendController 类的deleteProfileTrend 方法使用@RequiresAuthentication 注解确保用户已登录,再根据请求参数中的动态id去依次调用TrendMapper 中有关的删除方法;核心代码如下:

```
/**
 * 根据主键删除动态
 */
@Delete("DELETE FROM trend WHERE trend.id= #{trendId}")
int deletePostedTrend(@Param("trendId") Integer
trendId);

/**
```

```
 8    * 删除点赞表中的有关动态id的点赞
 9    */
10   @Delete("DELETE FROM likes WHERE likes.trend_id = #
     {trendId}")
11   int deletePostedTrendLikes(@Param("trendId") Integer
     trendId);
12   /**
13    * 删除评论表中的有关动态id的评论
14    */
15   @Delete("DELETE FROM comments WHERE comments.trend_id =
     #{trendId} ")
16   int deletePostedTrendComm(@Param("trendId") Integer
     trendId);
```

```
1  @DeleteMapping("/profile")
2  @RequiresAuthentication
3  @ApiOperation("删除档案中的动态")
4  public R deleteProfileTrend(@RequestParam("trendId")
   Integer trendId){
5      trendMapper.deletePostedTrend(trendId);
6      trendMapper.deletePostedTrendLikes(trendId);
7      trendMapper.deletePostedTrendComm(trendId);
8      return  R.success(null);
9  }
```

点赞关注模块 kun

点赞动态

主要由LikeController 类和LikesMapper 接口实现, LikesMapper 中 increaseTrendLikesCount 方法更新 动态表中的like_count字段,先查询动态的原like_count字段将其作为中间表,再在结果上+1 执行更新语句即完成了点赞数的更新;LikeController 类中likeTrend 方法,先从请求的JSON数据中取出trendId,再从请求头中获取userId,接着先查询是否已经点过

赞,构建Like 的通用查询用例,设置查询条件为'trendId'等于获取的trendId且'userId'等于获取的userId,如果LikesMapper 中selectCountByExample 方法返回数>0 则说明已经点赞过 直接返回失败提示信息, 为0 则创建新点赞记录插入到点赞表中,最后更新动态表中的字段;核心代码如下:

```
/**
 * 点赞动态  更新动态表中点赞数字段
 */
@Update("UPDATE trend " +
        "SET trend.likes_count=(SELECT likes_count FROM ( " +
        "SELECT trend.likes_count FROM trend WHERE trend.id=#{trendId}) AS t)+1 WHERE trend.id=#{trendId}")
int increaseTrendLikesCount(@Param("trendId")Integer trendId);
```

```
@PostMapping("/trend")
@RequiresAuthentication
@ApiOperation("点赞动态")
public R likeTrend(@RequestBody Map<String,Object> trendIdJson) {
    Integer trendId = (Integer) trendIdJson.get("trendId");
    // 先查询是否点赞过
    Subject subject = SecurityUtils.getSubject();
    Integer userId = JwtUtils.getUserId(subject.getPrincipals().toString());

    Example example = new Example(Likes.class);
    Example.Criteria criteria = example.createCriteria();
    criteria.andEqualTo("trendId", trendId).andEqualTo("userId",userId);
```

```
13      if (likesMapper.selectCountByExample(example) >0){
14          R.failed("已经点赞过咯哦");
15      }
16      // 新增点赞
17      Likes newLikes = new Likes();
18      newLikes.setTrendId(trendId);
19      newLikes.setUserId(userId);
20      newLikes.setCommentId(0);
21     likesMapper.insert(newLikes);
22      //更新动态中的点赞数
23      return R.success(
    likesMapper.increaseTrendLikesCount(trendId));
24  }
```

点赞评论

主要由LikeController 类 和 LikeMapper 接口实现.点赞评论与点赞动态类似,区别只是将操作表换成了评论表;核心代码如下:

```
1   /**
2    * 点赞评论 更新评论表中点赞数字段
3    */
4   @Update("UPDATE comments " +
5           "SET comments.likes_count=(SELECT likes_count
    FROM ( " +
6           "SELECT comments.likes_count FROM comments WHERE
    comments.id=#{commId}) AS t)+1 WHERE comments.id=#
    {commId}")
7   int increaseCommLikesCount(@Param("commId")Integer
    commId);
```

```
1    @PostMapping("/comm")
2    @RequiresAuthentication
3    @ApiOperation("点赞评论")
```

```
4   public R likeComm(@RequestBody Map<String,Object>
    commIdJson) {
5       Integer commId = (Integer)
    commIdJson.get("commId");
6       Integer trendId = (Integer)
    commIdJson.get("trendId");
7       // 先查询是否点赞过
8       Subject subject = SecurityUtils.getSubject();
9       Integer userId =
    JwtUtils.getUserId(subject.getPrincipals().toString());
10
11      Example example = new Example(Likes.class);
12      Example.Criteria criteria =
    example.createCriteria();
13      criteria.andEqualTo("commentId",
    commId).andEqualTo("userId",userId);
14      if (likesMapper.selectCountByExample(example) >0){
15          R.failed("已经点赞过咯哦");
16      }
17      // 新增点赞
18      Likes newLikes = new Likes();
19      newLikes.setTrendId(trendId);
20      newLikes.setUserId(userId);
21      newLikes.setCommentId(commId);
22      likesMapper.insert(newLikes);
23      //更新动态中的点赞数
24      return
    R.success(likesMapper.increaseCommLikesCount(commId));
25  }
```

取消点赞动态

主要由LikeController 类 和 LikeMapper 接口实现. 取消点赞与点赞类似, 区别在更新字段时为-1以前请求的方法为DELETE 核心代码如下:

```
/**
 * 取消点赞动态 更新动态表中点赞数字段
 */
@Update("UPDATE trend " +
        "SET trend.likes_count=(SELECT likes_count FROM ( " +
        "SELECT trend.likes_count FROM trend WHERE trend.id=#{trendId}) AS t)-1 WHERE trend.id=#{trendId}")
int decreaseTrendLikesCount(@Param("trendId")Integer trendId);
```

```
@DeleteMapping("/trend")
@RequiresAuthentication
@ApiOperation("取消点赞动态")
public R cancelLikeTrend(@RequestParam("trendId") Integer trendId) {
    // 先查询是否点赞过
    Subject subject = SecurityUtils.getSubject();
    Integer userId = JwtUtils.getUserId(subject.getPrincipals().toString());

    Example example = new Example(Likes.class);
    Example.Criteria criteria = example.createCriteria();
    criteria.andEqualTo("trendId", trendId).andEqualTo("userId",userId);

    // 删除点赞
    likesMapper.deleteByExample(example);
    //更新动态中的点赞数
    return R.success(likesMapper.decreaseTrendLikesCount(trendId));
}
```

取消点赞评论

与取消点赞动态类似,区别是把将操作表换为了评论表.核心代码如下:

```
/**
 * 取消点赞评论 更新评论表中点赞数字段
 */
@Update("UPDATE comments " +
        "SET comments.likes_count=(SELECT likes_count FROM ( " +
        "SELECT comments.likes_count FROM comments WHERE comments.id=#{commId}) AS t)-1 WHERE comments.id=#{commId}")
int decreaseCommLikesCount(@Param("commId")Integer commId);
```

```
@DeleteMapping("/comm")
@RequiresAuthentication
@ApiOperation("取消点赞评论")
public R cancelLikeComm(@RequestParam("commId") Integer commId) {
    Subject subject = SecurityUtils.getSubject();
    Integer userId = JwtUtils.getUserId(subject.getPrincipals().toString());

    Example example = new Example(Likes.class);
    Example.Criteria criteria = example.createCriteria();
    criteria.andEqualTo("commentId", commId).andEqualTo("userId",userId);

    // 删除点赞记录
    likesMapper.deleteByExample(example);
    //更新动态中的点赞数
```

```
15      return R.success(likesMapper.decreaseCommLikesCount(commId));
16  }
```

关注

关注与点赞类似,区别在于将被操作表换为了用户表.不过需要同时更新关注者的关注数和被关注者的被关注数,且增加了不能关注自己的判断
核心代码如下:

```
1   /**
2    * 增加用户关注数
3    */
4   @Update("UPDATE users " +
5           "SET users.following=( " +
6           "SELECT following FROM ( " +
7           "SELECT users.following FROM users WHERE users.id=#{userId}) AS t)+1 WHERE users.id=#{userId}")
8   int increaseUserFollowing(@Param("userId") Integer userId);
9
10   /**
11   * 增加用户的被关注数
12   */
13  @Update("UPDATE users " +
14              "SET users.followed=(SELECT followed FROM ( " +
15              "SELECT users.followed FROM users WHERE users.id=#{beFollowedUserId}) AS t)+1 WHERE users.id=#{beFollowedUserId}")
16  int increaseUserFollowed(@Param("beFollowedUserId") Integer beFollowedUserId);
```

```
1   @PostMapping
```

```
@RequiresAuthentication
@ApiOperation("关注其他用户")
public R followingOthers(@RequestParam("followingUserId") Integer followingUserId){
    Subject subject = SecurityUtils.getSubject();
    Integer userId = JwtUtils.getUserId(subject.getPrincipals().toString());
    if (followingUserId.equals(userId)){
        return R.failed("不能关注自己哦");
    }
    Follow follow = new Follow();
    follow.setUserId(userId);
    follow.setFollowingUserId(followingUserId);
    if (followMapper.selectOne(follow)!=null){
        return R.failed("不能重复关注哦");
    }
    followMapper.insert(follow);
    followMapper.increaseUserFollowing(userId);
    followMapper.increaseUserFollowed(followingUserId);
    return R.success(null);
}
```

取消关注

与关注类似 核心代码如下:

```
/**
 * 减少用户关注数
 */
@Update("UPDATE users " +
        "SET users.following=( " +
        "SELECT following FROM ( " +
        "SELECT users.following FROM users WHERE users.id=#{userId}) AS t)-1 WHERE users.id=#{userId}")
```

```
int decreaseUserFollowing(@Param("userId") Integer userId);


/**
 * 减少用户的被关注数
 */
@Update("UPDATE users " +
        "SET users.followed=(SELECT followed FROM ( " +
        "SELECT users.followed FROM users WHERE users.id=#{beFollowedUserId}) AS t)-1 WHERE users.id=#{beFollowedUserId}")
int decreaseUserFollowed(@Param("beFollowedUserId") Integer beFollowedUserId);
```

```
@DeleteMapping
@RequiresAuthentication
@ApiOperation("取消关注其他用户")
public R cancelFollowingOthers(@RequestParam("cancelFollowingUserId") Integer cancelFollowingUserId){
    Subject subject = SecurityUtils.getSubject();
    Integer userId = JwtUtils.getUserId(subject.getPrincipals().toString());
    Follow follow = new Follow();
    follow.setUserId(userId);
    follow.setFollowingUserId(cancelFollowingUserId);
    if (followMapper.selectOne(follow)==null){
        return R.failed("不能重复取消关注哦");
    }
    followMapper.delete(follow);
    followMapper.decreaseUserFollowing(userId);
```

```
15
     followMapper.decreaseUserFollowed(cancelFollowingUserI
    d);
16      return R.success(null);
17  }
```

获取关注列表

由FollowMapper 接口和 FollowController 实现.FollowMapper 中 getUserFollowing 方法查询在关注表中user_id 为参数的子查询集合中的用户.FollowController 中 getFollowingUsers 方法则 根据请求参数调用 FollowMapper 中getUserFollowing 方法.核心代码如下:

```
1   /**
2    * 获取某个用户的关注列表
3    */
4   @Select("SELECT u.id,u.user_name
    userName,u.signature,u.avatar  FROM users u WHERE u.id
    IN ( " +
5           "SELECT f.following_user_id FROM follow f WHERE
    f.user_id=#{userId})")
6   List<Users> getUserFollowing(@Param("userId") Integer
    userId);
```

```
1   @GetMapping("/ing")
2   @ApiOperation("获取其他用户的关注列表")
3   public R<List<Users>>
    getFollowingUsers(@RequestParam("userId") Integer
    userId){
4       return
    R.success(followMapper.getUserFollowing(userId));
5   }
```

获取被关注列表

与获取关注列表类似.核心代码如下:

```
/**
 * 获取某个用户的被关注列表
 */
@Select("SELECT u.id,u.user_name userName,u.signature,u.avatar  FROM users u WHERE u.id IN ( " +
        "SELECT f.user_id FROM follow f WHERE f.following_user_id=#{userId})")
List<Users> getUserFollowed(@Param("userId") Integer userId);
```

```
@GetMapping("/ed")
@ApiOperation("获取其他用户的被关注列表")
public R<List<Users>> getFollowedUsers(@RequestParam("userId") Integer userId) {
    return R.success(followMapper.getUserFollowed(userId));
}
```

评论模块 ding

获取评论

主要由CommentController 类和 CommentService接口实现.CommentService 中getCommByTrendId 方法 由注入的CommentsMapper中的getCommByTrendId 方法实现,该方法将评论表和用户表根据评论表中user_id字段等于用户表id字段且评论表trend_id字段等于参数的规则查询评论;CommentController 类中的getComm 方法根据请求参数中的trendId 调用 CommentService 中的getCommByTrendId() 返回一个CommentDto 集合.核心代码如下:

```
/**
 * 根据动态id获取评论
 */
@Select("SELECT u.id userId,u.user_name userName,u.avatar userAvatar,c.create_time createTime,c.likes_count likesCount,c.content,c.images,c.id commId,c.parent_id parentId FROM comments AS c INNER JOIN users AS u ON c.user_id=u.id AND c.trend_id=#{id} ORDER BY create_time desc")
@ResultType(CommentDto.class)
List<CommentDto> getCommByTrendId(Integer trendId);
```

```
@GetMapping
@ApiOperation("获取动态评论")
public R<List<CommentDto>> getComm(@RequestParam("trendId") Integer trendId){
    return R.success(commentService.getCommByTrendId(trendId));
}
```

发布评论

主要由CommentController 类和 CommentService接口实现. CommentService 中increaseCommentCount 方法 先根据参数查询所属动态的原comments_count字段将其作为中间表,再在结果上+1 执行更新语句即完成了评论数的更新.CommentController 类中的postComment 方法则是先将POST请示的JSON格式的字符串转为Comments 对象,再从请求头Authentication 中获取用户id,为新评论设置创建时间以及userId,使用 commentService 的 insertSelective 方法插入新评论,为null的属性将会使用数据库默认值,最后更新动态的评论数. 核心代码如下:

```java
/**
 * 动态评论数增加
 */
@Update("UPDATE trend " +
        "SET trend.comments_count=(SELECT comments_count FROM (" +
        "SELECT trend.comments_count FROM trend WHERE trend.id=#{trendId}) AS t)+1 WHERE trend.id=#{trendId}")
int increaseCommentCount(@Param("trendId")Integer trendId);
```

```java
@PostMapping
@RequiresAuthentication
@ApiOperation("发布评论")
public R postComment(@RequestBody String postCommJson){
    // 添加新评论
    Subject subject = SecurityUtils.getSubject();
    Integer userId = JwtUtils.getUserId(subject.getPrincipals().toString());
    Comments comments = JSON.parseObject(postCommJson, Comments.class);
    comments.setCreateTime(new Date());
    comments.setUserId(userId);
    logger.info(" new Comment" + comments.toString());
    commentService.insertSelective(comments);
    // 更新动态评论值

    trendService.increaseCommentCount(comments.getTrendId());
    return R.success(null);
}
```

技术总结

此项目前端使用的技术栈为 Vue Vant UI Axios Vuex Vue router

- 通过 Vue 驱动 JS 引擎 对视图数据双向绑定(MVVM)
- 使用 Vant UI 组件库 快速搭建页面交互原型
- 使用 Axios 异步请求后端接口
- 使用 Vuex 对项目中使用频度高的的信息(如 token)进行缓存
- 使用 Vue router 管理调度项目所有页面

此项目后端使用的技术栈为 SpringBoot tkMybatis Shiro Jwt

- 使用 SpringBoot 进行微服务快速搭建
- 使用tkMybatis 简化开发
- Shiro 配合 Jwt 实现认证鉴权管理
