我的个人博客设计说明书

Author:凡朝剑 Date:20200901

一、实体类设计

1. 登录用户blogUser

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 字段 | 类型 | 说明 |
| 主键 | userId | Integer |  |
| 注册方式 | registerMethod | String | Localhost、qq、wechat。通过不同的注册方式获取不同的用户信息 |
| 昵称 | name | String |  |
| 邮箱 | email | String |  |
| Qq登录用户 | openId | String | 使用第三方QQ登录的用户 |
| 微信登录用户 | unionId | String | 使用第三方微信登录的用户 |
| 头像 | icon | String | 如果是本地注册的用户系统会自动选择一张预设的图片作为头像 |
| 性别 | gender | String |  |
| 邮件回复 | emailReply | Integer |  |
| 是否启用 | isInuse | Integer | 1：启用。0：没有启用，可以看成是账号已注销 |

1. 管理员用户BlogAdmin

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 字段 | 类型 | 说明 |
| 主键 | adminId | Integer |  |
| 用户名 | username | String |  |
| 密码 | password | String |  |
| 昵称 | name | String |  |
| 职业 | job | String |  |
| 邮箱 | email | String |  |
| 籍贯 | place | String |  |
| 爱好 | hobby | String |  |
| 格言 | motto | String |  |
| 简介地址 | introUrl | String |  |
| 性别 | gender | String |  |
| 头像 | iconUrl | String |  |
| Qq二维码 | qqQrcode | String |  |
| 微信二维码 | wechatQrcode | String |  |
| 微博二维码 | weboQrcode | String |  |
| 公众号二维码 | gghQrcode | String |  |
| Github地址 | githubUrl | String |  |
| 排序 | preOrder | Integer | 排序，用于随意切换网站所有人 ASC ，取第一个为网站的管理员 |
| 站点 | SiteId |  | 外键，与“站点信息表是一对一的关系” |

1. 站点信息SiteInfo

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 字段 | 类型 | 说明 |
| 主键 | siteId | Integer | Auto\_increment |
| 站点名称 | siteName | String |  |
| 站点logo | logoImg | String |  |
| 站点背景 | siteBgiUrl | String |  |
| 站点简介 | siteIntroUrl | String |  |
| 站点二维码 | siteQrcode | String |  |
| 站龄 | siteAge | String |  |
| 访客 | visitorCount | Integer |  |
| 文章总数 | articleCount | Integer |  |

1. 博客文章Article

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 字段 | 类型 | 说明 |
| 主键 | articleId | Integer |  |
| 文章标题 | title | String |  |
| 作者 | author | String |  |
| 简介图片 | iconUrl | String |  |
| 文章内容 | content | String |  |
| 创建时间 | createDate | Date |  |
| 留言数量 | commentCount | Integer |  |
| 阅览量 | readCount | Integer |  |
| 获赞数 | likeCount | Integer |  |
| 转载地址 | reprintSite | String |  |
| 标签分类 | labelId |  | 外键，和“标签表”是多对一关系 |

1. 心情随笔Essay

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 字段 | 类型 | 说明 |
| 主键 | essayId | Integer |  |
| 创建时间 | createDate | Date |  |
| 内容 | content | String |  |

1. 标签表Label

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 字段 | 类型 | 说明 |
| 主键 | labelId | Integer |  |
| 名称 | labelName | String |  |
| 文章数量 | articleCount | Integer |  |
| 标签说明 | remark | String |  |
| 创建时间 | createDao | Date |  |

1. 评论表Comment

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 字段 | 类型 | 说明 |
| 主键 | id | Integer |  |
| 评论类型 | type | String |  |
| 文章id | articleId | Integer |  |
| 评论内容 | content | String |  |
| 评论时间 | createDate | Date |  |
| 用户id | userId |  | 外键，与用户表是多对一的关系 |

1. 回复表Reply

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 字段 | 类型 | 说明 |
| 主键 | Id | Integer |  |
| 回复目标 | targetId | Integer | replyId,commentId。回复的目标可能是评论的回复也可能是回复的回复 |
| 文章id | articleId | Integer | 文章的id，这儿不做外键处理 |
| 回复类型 | replyType | String | 两种，1：对评论的回复comment。2：对回复的回复reply |
| 回复内容 | content | String |  |
| 创建时间 | createDate | Date |  |
| 评论id | commentId |  | 外键，评论表 多对一关系 |
| 回复用户id | fromUid |  | 外键，用户 多对一关系 |
| 目标用户id | toUid |  | 外键，用户 多对一关系 |

1. 接口及说明·
2. Article

* 接口名称：article

功能：查找所有的博客文章列表

参数：pageSize，pageCount

请求方式：GET

返回值：



* 接口名称：article/findByLabel

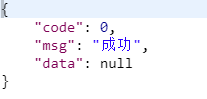
功能：通过标签名查找所有的文章列表

参数：labelName 标签名称

请求方式：GET

返回值：

1. 当没有查找到相应的标签时



1. 正常情况



* 接口名称article/baseAdmin/save

权限：baseAdmin

功能：保存博客文章

参数：labelName，title，author，iconUrl，intro，content，reprintSite

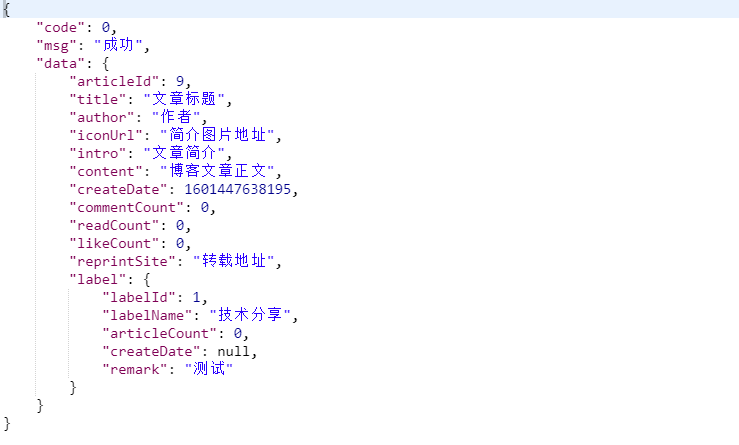
请求方式：POST

返回值：

1. 没有标签名为labelName的标签时



1. 正常保存



* 接口名称article/baseAdmin/delete

权限：baseAdmin

功能：删除指定文章id的文章

参数：articleId

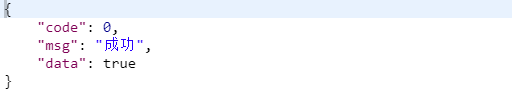
请求方式：POST

返回值：

1. 没有找到指定articleId的记录时



1. 正常情况下



* 接口名称 article/baseAdmin/update

权限：baseAdmin

功能：修改博客文章基本信息

参数：articleId (文章id，必填)，title，author，iconUrl，intor，content，reprintSite

请求方式：POST

返回值：

1. 没有指定articleId参数



2、没有指定articleId的文章



3、正常情况



* 接口名称 article/dynamicInfo

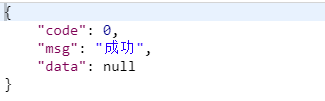
功能：修改文章动态字段：如评论数、浏览量、获赞数

参数：articleId、type(comment：文章评论、read：浏览、like：获赞)

请求方式：POST

返回结果：

1. 正常返回



1. BlogAdmin

* 接口名称：blogAdmin

功能：查找管理员用户（默认查找按照preOrder字段ASC排序的第一条记录）

参数：无

请求方式：GET

返回结果：



* 接口名称 blogAdmin/baseAdmin/updateBaseInfo

权限：baseAdmin

功能：根据指定的adminId更改信息

参数：adminId、name、job、email、place、hobby、motto、introUrl、gender、iconUrl、qqQrcode、wechatQrcode、weboQrcode、gghQrcode、githubUrl

请求方式：POST

返回值：



* 接口名称 blogAdmin/baseAdmin/updateLoginInfo

功能：修改指定adminId管理员的用户名和密码

参数：adminId、username、password

请求方式：POST

返回值：

1. 正常情况



* 接口名称 blogAdmin/login

功能：登录逻辑

参数：username、password

请求方式：POST

返回值:



1. Comment

* 接口名称 comment

功能：查找指定articleId的评论，以及评论下的所有回复

参数：articleId

请求方式：GET

返回值：





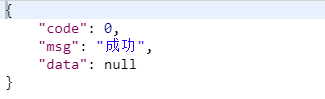
* 接口名称 comment/{commentId}

功能：通过commentId查找所有的评论及回复

参数：path /{commentId}

返回值：

1. 评论此commentId的评论



1. 正常情况

【同 通过articleId获取所有的评论及回复】

* 添加评论 comment/baseUser/addComment

权限：baseUser

功能：添加评论

参数：type（comment：评论，Guestbook：留言）、articleId、userId、content

请求方式：POST

请求结果：



* 接口名称comment/baseSelf/deleteComment

权限：baseSelf

功能：删除评论及其关联的所有回复

参数：userID、commentID

返回值：

。。。。。。

1. Essay

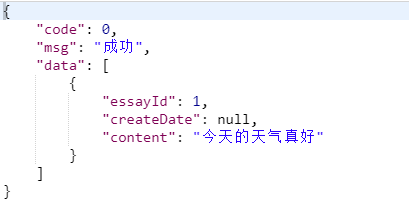
* 接口名称 essay/

功能：查找所有心情随笔

参数：无

请求方式：GET

返回值：



* 接口名称 essay/baseAdmin/delete

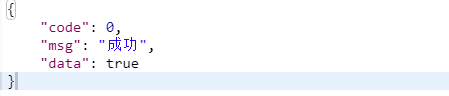
权限：baseAdmin

功能：删除指定id的随笔记录

参数：id

请求方式：POST

返回值：

=

* 接口名称essay/baseAdmin/save

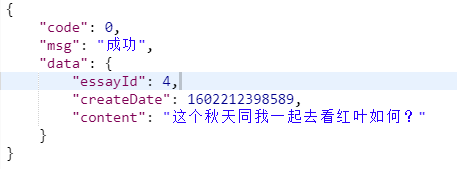
权限：baseAdmin

功能：添加随笔

参数：content

请求方式：POST

返回值：



1. Label

* 接口名称label/findByName

功能：通过名称查找标签

参数：labelName

请求方式：GET

返回值：



* 接口名称label/

功能：查找所有的标签

参数：无

请求方式：GET

返回值：



* 接口名称label/baseAdmin/update/

功能：修改标签信息

参数：labelid、labelName、remark

请求方式：POST

返回值：返回修改过后的标签



* 接口名称label/baseAdmin/save

权限：baseAdmin

功能：添加标签

参数：labelName、remark

请求方式：POST

返回值：返回新添加的标签



1. Reply-

* 接口名称reply/baseUser/add

权限：baseUser

功能：添加回复

参数：articleID、commentID、targetID、formUid、toUid、replyType、content

请求方式：POST

返回值：返回回复的数据



1. User

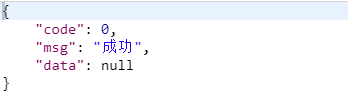
* 接口名称user/login

功能：用户登录（本地注册的用户）

参数：name、email

请求方式：POST

返回值：返回用户基本信息，或者NULL





* 接口名称user/baseSelf/modifyInfo

权限：baseSelf

功能：修改用户基本信息

参数：userID、email、icon、gender、emailReply

请求方式：POST

返回值：修改后的用户



* 接口名称user/baseAdmin/getAll

权限：baseAdmin

功能：查找所有用户

参数：无

请求方式：GET

返回值：

。。。。

* 接口名称user/addOrFind

功能：添加或查找用户，如果是本地注册的账户为添加，第三方登录的账户添加或查找，有则查找，无则添加

参数：registerMethod、name、email、openId（取决于registerMethod为“qq”时）、unionId（取决于registerMethod为“wechat”时）、icon、gender、emailReply

请求方式：POST

返回结果：

。。。。。。

* 接口名称user/baseSelf/findById

权限：baseSelf

功能：通过id查找用户

参数：userID

请求方式：GET

返回结果：



* 接口名称user/baseAdmin/dop

权限：baseAdmin

功能：销毁账户并删除与之相关联的所有评论与回复（太过暴力）

参数：userID

请求方式：POST

返回值：

。。。。。

1. 关于平台接口访问权限的处理

* 平台接口的访问权限一共设有4个

1. 供系统管理员访问（baseAdmin）

场景：所有平台接口

1. 供登录用户访问（baseUser）

场景：参与用户评论与回复

1. 供登录用户自己访问（baseSelf）

查看和修改自己的信息，删除自己的评论与回复，注销自己账号

1. 所有人可访问

* 访问接口的方式

1. 系统管理员

访问方式：在header中添加认证信息

Key：BaseAdmin

Value：Base64(用户名&密码)

说明：通过Base64编码的登录信息

1. 登录用户

访问方式：在header中添加认证信息

Key：BaseUser

Value：

1. 本地注册用户（localhost）

Base64(localhost#name&email)

1. 第三方QQ登录用户（qq）

Base64(qq#openId)

1. 第三方微信登录用户（wechat）

Base64(wechat#unionId)

1. 登录用户自己

访问方式：在header中添加认证信息

Key：BaseSelf

Value:

。。。。。。【同上】

* 权限处理（拦截器）

1. 拦截方式

请求地址拦截，三种需要权限的接口地址分别包含baseAdmin、baseUser、baseSelf

1. 主要代码

**package** com.fanchaojian.utils;  
**import** com.alibaba.fastjson.JSON;  
**import** com.fanchaojian.domain.BlogAdmin;  
**import** com.fanchaojian.domain.User;  
**import** com.fanchaojian.service.IBlogAdminService;  
**import** com.fanchaojian.service.IUserService;  
**import** com.mysql.jdbc.util.Base64Decoder;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.util.Base64Utils;  
**import** org.springframework.web.servlet.HandlerInterceptor;  
**import** org.springframework.web.servlet.ModelAndView;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.IOException;  
**import** java.io.PrintWriter;  
**import** java.util.Base64;  
*/\*\*  
 \* 用于登录拦截的过滤器  
 \* 平台所有的接口一共有三种不用的权限访问机制  
 \* 1：baseAdmin：系统管理员，可访问平台所有的接口  
 \* 2：baseUser：平台注册用户，可操作可供允许的平台注册用户所有的数据（如评论，回复功能）  
 \* 3：baseSelf：只能操作自己的数据（如查看和修改个人信息，删除自己的评论以及回复）  
 \*/***public class** LoginInterceptor **implements** HandlerInterceptor {  
 @Autowired  
 **private** IBlogAdminService **blogAdminService** ;  
 @Autowired  
 **private** IUserService **userService** ;  
 @Override  
 **public boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object o) **throws** Exception {  
 String servletPath = request.getServletPath();  
 String result = JSON.*toJSONString*(ResultUtils.*error*(101, **"登录认证失败"**));  
 *//判断请求路径中是否包含指定的字符串save,delete,update,get* **if**(servletPath.contains(**"baseAdmin"**)){  
 BlogAdmin blogAdmin = adminAuthorization(request,response,**"BaseAdmin"**) ;  
 **if**(blogAdmin == **null**){  
 returnJson(response, result);  
 **return false** ;  
 }**else**{  
 **return true** ;  
 }  
 }**else if**(servletPath.contains(**"baseUser"**)){  
 BlogAdmin blogAdmin = adminAuthorization(request,response,**"BaseAdmin"**) ;  
 User user = userAuthorization(request,response,**"BaseUser"**) ;  
 **if**(blogAdmin != **null** || user != **null**){  
 **return true** ;  
 }**else**{  
 returnJson(response, result);  
 **return false** ;  
 }  
 }**else if**(servletPath.contains(**"baseSelf"**)){  
 BlogAdmin blogAdmin = adminAuthorization(request,response,**"BaseAdmin"**) ;  
 User user = userAuthorization(request,response,**"BaseSelf"**) ;  
 **if**(blogAdmin != **null**){  
 **return true** ;  
 }**else if**(user != **null**){  
 *//获取header中的userID参数* **int** userID = Integer.*parseInt*(request.getParameter(**"userID"**)) ;  
 User userById = **userService**.findById(userID);  
 **if**(user.getUserId() == userById.getUserId()){  
 **return true** ;  
 }**else**{  
 returnJson(response, result);  
 **return false** ;  
 }  
 }**else**{  
 returnJson(response, result);  
 **return false** ;  
 }  
 }**else**{  
 **return true** ;  
 }  
 }  
 **private** BlogAdmin adminAuthorization(HttpServletRequest request, HttpServletResponse response,String headerStr){  
 String authorization = request.getHeader(headerStr);  
 **if**(authorization == **null**){  
 **return null** ;  
 }  
 String authText = **""** ;  
 *//base64解码* Base64.Decoder decoder = Base64.*getDecoder*();  
 **try**{  
 authText = **new** String(decoder.decode(authorization), **"UTF-8"**);  
 }**catch**(Exception e){  
 **return null** ;  
 }  
  
 **if**(!authText.matches(**"^\\w+&{1}\\w+$"**)){  
 **return null** ;  
 }  
 String username = authText.split(**"&"**)[0] ;  
 String password = authText.split(**"&"**)[1] ;  
 System.***out***.println(**"用户名："**+username+**",密码："**+password) ;  
 BlogAdmin admin = **blogAdminService**.login(username, password);  
 **return** admin ;  
 }  
 **private** User userAuthorization(HttpServletRequest request,HttpServletResponse response,String headerStr) {  
 String authorization = request.getHeader(headerStr) ;  
 **if**(authorization == **null**){  
 **return null** ;  
 }  
 *//base64 解码* String authText = **""** ;  
 *//base64解码* Base64.Decoder decoder = Base64.*getDecoder*();  
 **try**{  
 authText = **new** String(decoder.decode(authorization), **"UTF-8"**);  
 }**catch**(Exception e){  
 **return null** ;  
 }  
 *//localhost#admin&1948556024@qq.com qq#openId wechat#unionId* **if**(!authText.matches(**"^\\w+#{1}\\w+&?(\\w+([-+.]\\w+)\*@\\w+([-.]\\w+)\*\\.\\w+([-.]\\w+)\*)?$"**)){  
 **return null** ;  
 }  
 String registerMethod = authText.split(**"#"**)[0] ;  
 **if**(registerMethod.equals(**"localhost"**)){  
 *//获取name和email，并从数据库查找* String authContext = authText.split(**"#"**)[1] ;  
 *//admin&1948556024@qq.com* **if**(!authContext.matches(**"^\\w+&{1}\\w+([-+.]\\w+)\*@\\w+([-.]\\w+)\*\\.\\w+([-.]\\w+)\*$"**)){  
 **return null** ;  
 }  
 String name = authContext.split(**"&"**)[0] ;  
 String email = authContext.split(**"&"**)[1] ;  
 *//通过name和email查找用户* User user = **userService**.login(name, email);  
 **return** user ;  
 }**else if**(registerMethod.equals(**"qq"**)){  
 String openId = authText.split(**"#"**)[1] ;  
 **if**(!openId.matches(**"^\\w+$"**)){  
 **return null** ;  
 }  
 *//通过openID查找用户* User UserByOpenId = **userService**.findByOpenId(openId);  
 **return** UserByOpenId ;  
  
 }**else if**(registerMethod.equals(**"wechat"**)){  
 String unionId = authText.split(**"#"**)[1] ;  
 **if**(!unionId.matches(**"^\\w+$"**)){  
 **return null** ;  
 }  
 *//通过unionId查找用户* User userByUnionId = **userService**.findByUnionId(unionId);  
 **return** userByUnionId ;  
 }**else**{  
 **return null** ;  
 }  
  
 }  
 **private void** returnJson(HttpServletResponse response, String result) {  
 PrintWriter writer = **null**;  
 response.setCharacterEncoding(**"UTF-8"**);  
 response.setContentType(**"text/html; charset=utf-8"**);  
 **try** {  
 writer = response.getWriter();  
 writer.print(result);  
  
 } **catch** (IOException e) {  
 } **finally** {  
 **if** (writer != **null**) {  
 writer.close();  
 }  
 }  
 }  
  
 @Override  
 **public void** postHandle(HttpServletRequest request, HttpServletResponse response, Object o, ModelAndView modelAndView) **throws** Exception {  
 }  
  
 @Override  
 **public void** afterCompletion(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Object o, Exception e) **throws** Exception {  
 }  
}

1. 1
2. 1
3. 1
4. 1
5. 1
6. 1