项目内容

实现博客网站，有登录，注册，查看文章，发布文章，评论文章，点赞文章和个人信息的功能，并且用户区分管理员，普通用户和游客用户，游客用户只能查看文章，不能发布文章。而且只有管理员账号可以进入后台页面，查看所有的用户信息，文章信息，评论信息，用户的登录登出信息，后端 `sql` 语句的执行日志信息和用户身份的配置信息。并且页脚记录了网站的访问量。

项目的技术栈

本项目采用前后端分离的架构，前端使用 `Vue` 框架编写，后端使用 `SpringBoot` 框架加上 `MyBatis` 数据库操作框架编写。用 `MySQL` 数据库来存放用户，文章，评论信息，用 `redis` 数据库来存取网站的访问量。

数据库的设计

一共有六个：
`tb_user` 负责记录用户信息。有 `id` ， `username` ， `password` （进行 `md5` 加密）， `type` （用户身份）， `avatar` （用户头像）等用户的信息数据。

```
+---------------+---------------------------------+------+-----+---------+----------------+
| Field         | Type                            | Null | Key | Default | Extra          |
+---------------+---------------------------------+------+-----+---------+----------------+
| id            | int(11)                         | NO   | PRI | NULL    | auto_increment |
| username      | varchar(30)                     | NO   | UNI | NULL    |                |
| password      | varchar(300)                    | NO   |     | NULL    |                |
| type          | enum('admin','user','visitor') | NO   |     | user    |                |
| phone         | varchar(20)                     | YES  |     |         |                |
| email         | varchar(30)                     | YES  |     |         |                |
| age           | int(11)                         | YES  |     | NULL    |                |
| sex           | varchar(3)                      | YES  |     | 0       |                |
| location      | varchar(100)                    | YES  |     |         |                |
| avatar        | varchar(1000)                   | YES  |     |         |                |
| last_login    | datetime                        | YES  |     | NULL    |                |
| register_time | datetime                        | NO   |     | NULL    |                |
| is_delete     | tinyint(4)                      | YES  |     | 0       |                |
| ps            | text                            | YES  |     | NULL    |                |
| intro         | text                            | YES  |     | NULL    |                |
+---------------+---------------------------------+------+-----+---------+----------------+
15 rows in set (0.03 sec)
```

`tb_passage` 负责记录文章信息，包括文字内容，文章作者信息，文章分类标签，文章标题，文章的点赞数，文章的访问数，文章的最近更新时间，文章的发布时间。

```
mysql> show columns from tb_passage;
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| id          | int(11)     | NO   | PRI | NULL    | auto_increment |
| author_id   | int(11)     | NO   |     | NULL    |                |
| description | text        | NO   |     | NULL    |                |
| title       | varchar(50) | NO   |     | NULL    |                |
| content     | text        | YES  |     | NULL    |                |
| type        | varchar(30) | NO   |     | NULL    |                |
| create_time | datetime    | NO   |     | NULL    |                |
| update_time | datetime    | YES  |     | NULL    |                |
| like_num    | int(11)     | NO   |     | 0       |                |
| visit_num   | int(11)     | NO   |     | 0       |                |
| is_delete   | tinyint(4)  | YES  |     | 0       |                |
+-------------+-------------+------+-----+---------+----------------+
11 rows in set (0.00 sec)
```

`tb_comment` 负责记录文章的评论信息，包括评论内容，评论的文章 `id` ，评论的用户信息，评论的点赞数，评论的发布时间。

```
mysql> show columns from tb_comment;
+--------------------+-------------+------+-----+---------+----------------+
| Field              | Type        | Null | Key | Default | Extra          |
+--------------------+-------------+------+-----+---------+----------------+
| id                 | int(11)     | NO   | PRI | NULL    | auto_increment |
| passage_id         | int(11)     | NO   |     | NULL    |                |
| content            | text        | NO   |     | NULL    |                |
| commentator_id     | int(11)     | NO   |     | NULL    |                |
| commentator_username | varchar(30) | NO   |     | NULL    |                |
| create_time        | datetime    | NO   |     | NULL    |                |
| like_num           | int(11)     | NO   |     | 0       |                |
| is_delete          | tinyint(4)  | YES  |     | 0       |                |
+--------------------+-------------+------+-----+---------+----------------+
8 rows in set (0.01 sec)
```

`tb_sql_log` 是用于记录 `sql` 语句的执行日志信息，用 `MySQL` 自带的触发器功能实现。

```
mysql> show columns from tb_sql_log;
+----------------+----------+------+-----+---------+----------------+
| Field          | Type     | Null | Key | Default | Extra          |
+----------------+----------+------+-----+---------+----------------+
| id             | int(11)  | NO   | PRI | NULL    | auto_increment |
| operation      | text     | NO   |     | NULL    |                |
| time           | datetime | NO   |     | NULL    |                |
| operation_params | text   | NO   |     | NULL    |                |
+----------------+----------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

`tb_login_log` 用于记录用户的登录登出日志，包括用户登录退出时的 `ip` 和时间。

```
mysql> show columns from tb_login_log;
+----------+---------------------------------+------+-----+---------+----------------+
| Field    | Type                            | Null | Key | Default | Extra          |
+----------+---------------------------------+------+-----+---------+----------------+
| id       | int(11)                         | NO   | PRI | NULL    | auto_increment |
| username | varchar(30)                     | NO   |     | NULL    |                |
| type     | enum('admin','user','visitor') | NO   |     | NULL    |                |
| ip       | varchar(20)                     | YES  |     |         |                |
| time     | datetime                        | NO   |     | NULL    |                |
| action   | enum('登录','退出')             | NO   |     | NULL    |                |
+----------+---------------------------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)

mysql>
```

`tb_about` 用于记录管理员对网页的介绍。

```
mysql> show columns from tb_about;
+---------+---------+------+-----+---------+----------------+
| Field   | Type    | Null | Key | Default | Extra          |
+---------+---------+------+-----+---------+----------------+
| id      | int(11) | NO   | PRI | NULL    | auto_increment |
| content | text    | NO   |     | NULL    |                |
+---------+---------+------+-----+---------+----------------+
2 rows in set (0.00 sec)
```

思路分析

在整体架构方面，后端采取了 `MVC` 的模式，将代码分为 `pojo` 层， `dao` 层和 `controller` 层。并且用 `service` 层封装了业务代码。

在登录方面，这里登录使用了 `JWT` 技术，通过 `JWT` 来防止客户端伪造 `admin` ，来保护网站的安全性。示例代码如下：

```
package com.just.utils;

import com.just.pojo.User;
import com.just.service.UserService;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import javax.servlet.http.HttpServletRequest;
import java.security.PrivateKey;
```

```java
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

@Component
public class JwtUtil {

    /**
     * Spring 容器所管理的主要是对象实例，而@Autowired 依赖注入的都是容器内的对象实例，
     * 在 Java 中 static 修饰的静态属性（变量和方法）是属于类的，而非属于实例对象。
     * 当类加载器加载静态变量时，Spring上下文尚未加载完成，则类加载器不会在 Bean
     * 中正确注入属性。如下是一个错误的例子：
     */
//    @Autowired
//    private static UserService userService;

    //办法1：@Autowired + 构造器
    private static UserService userService;

    @Autowired
    public JwtUtil(UserService userService) {
        JwtUtil.userService = userService;
    }

    //一天过期
    private static long expire = 60 * 60 * 24;
    //JWT密钥
    private static String secretKey =
"helloworld@Ilikeyou_justdoit+thestoryneverends~#lauv;ItsOK!~";

    /**
     * 提供username和type返回token
     *
     * @param username
     * @param type
     * @return
     */
    public static String generateToken(String username, String type) {
        Date now = new Date();
        Date expiration = new Date(now.getTime() + 1000 * expire);
        Map<String, Object> payload = new HashMap<>();
        payload.put("username", username);
        payload.put("type", type);
        return Jwts.builder()
                .setHeaderParam("type", "JWT")
                .setClaims(payload)
```

```
                .setIssuedAt(now)
                .setExpiration(expiration)
                .signWith(SignatureAlgorithm.HS512, secretKey)
                .compact();
    }


    /**
     * 验证token是否有效
     *
     * @param token
     * @param user
     * @return
     */
    public static boolean validateToken(String token, User user) {
        String username = getUsernameByToken(token);
        String type = getTypeByToken(token);
        Date expiredDate = getExpiredDateByToken(token);
        return username.equals(user.getUsername()) && type.equals(user.getType())
&& !expiredDate.before(new Date());
    }

    /**
     * 从token中获取用户名
     *
     * @param token
     * @return
     */
    public static String getUsernameByToken(String token) {
        return (String) getClaimsByToken(token).get("username");
    }

    /**
     * 从toke中获取用户类型
     *
     * @param token
     * @return
     */
    public static String getTypeByToken(String token) {
        return (String) getClaimsByToken(token).get("type");
    }

    /**
     * 从toke中获取到期时间
     *
     * @param token
```

```
     * @return
     */
    public static Date getExpiredDateByToken(String token) {
        return getClaimsByToken(token).getExpiration();
    }

    /**
     * 从token中获取claims
     * @param token
     * @return
     */
    public static Claims getClaimsByToken(String token) {
        return Jwts.parser()
                .setSigningKey(secretKey)
                .parseClaimsJws(token)
                .getBody();
    }

    /**
     * 从request获取用户id的工具方法，简化controller层的代码
     * @param request
     * @return
     */
    public static Integer getIdFromRequest(HttpServletRequest request) {
        try {
            String token = request.getHeader("Authorization");
            String username = JwtUtil.getUsernameByToken(token);
            User user = userService.selectByUsername(username);
            return user.getId();
        } catch (Exception e) {
            return null;
        }
    }

    /**
     * 从request获取用户type的工具方法，简化controller层验证身份时的代码
     * 也便于以后修改token的存储方式
     * @param request
     * @return
     */
    public static String getTypeFromRequest(HttpServletRequest request) {
        try {
            String token = request.getHeader("Authorization");
            return getTypeByToken(token);
        } catch (Exception e) {
            return null;
```

```
        }
    }
}
```

在 `dao` 层方面，通过 `MyBatis` 来操作数据库，并使用 `@Results` 注解，解决了数据库字段名和 `pojo` 实体类字段名不一致的问题。示例代码如下：

```
package com.just.dao;

import com.just.pojo.About;
import com.just.pojo.Comment;
import com.just.pojo.Passage;
import org.apache.ibatis.annotations.*;

import java.util.List;

@Mapper
public interface PassageDao {
    @Select("select * from tb_passage where is_delete != 1")
    @Results(
            id = "passageMap",
            value ={
                @Result(column = "author_id", property = "authorId"),
                @Result(column = "create_time", property = "createTime"),
                @Result(column = "update_time", property = "updateTime"),
                @Result(column = "like_num", property = "likeNum"),
                @Result(column = "visit_num", property = "visitNum"),
                @Result(column = "is_delete", property = "del"),
            }
    )
    public List<Passage> selectAll();

    @Select("select * from tb_passage where author_id = #{authorId} and is_delete
!= 1")
    @ResultMap({"passageMap"})
    public List<Passage> selectPassageByAuthorId(Integer authorId);

    @Select("select * from tb_passage where id = #{id} and is_delete != 1")
    @ResultMap({"passageMap"})
    public Passage selectById(Integer id);

    @Delete("update tb_passage set is_delete = 1 where id = #{id}")
    public int deletePassage(Integer id);

    @Update("update tb_passage set title = #{title}, content = #{content}," +
            "description = #{description},type = #{type}, update_time = #
```

```
{updateTime} where id = #{id}")
    public int updatePassage(Passage passage);

    @Insert("insert into tb_passage
(id,author_id,description,title,content,type,create_time) " +
            "values (null, #{authorId},#{description},#{title},#{content},#{type},#
{createTime})")
    public int add(Passage passage);

    @Select("select max(id) from tb_passage")
    public int selectMaxId();

    @Update("update tb_passage set like_num = like_num + 1 where id = #
{passageId}")
    public int increaseLikeNum(Integer passageId);

    @Select("select * from tb_about where id = 1")
    public About selectAbout();

    @Update("update tb_about set content = #{content} where id = 1")
    public int updateAbout(String content);

    @Update("update tb_passage set update_time = #{update_time} where id = #{id}")
    public int updateUpdateTime(Passage passage);
}
```

在记录网页访问量的时候，我使用了 `Spring` 中的 `AOP` 技术，来对全局路由访问记录访问量，这里其实还有待优化，因为用户的一次请求可能会经历多次访问接口。
`AOP` 切面代码：

```
package com.just.controller.aop;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.core.ValueOperations;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;

@Component
@Aspect
public class LogAdvice {
    @Resource
```

```
    private RedisTemplate redisTemplate;

    /**
     * 对所有代码进行aop，记录网站访问量
     * 但是弊端是一次请求会有多个数据包，导致记录多次
     * 暂时只记录主页的访问量
     */
    @Pointcut("execution(public *
com.just.controller.PassageController.selectIndexPassage(..))")
    private void pv(){}

    @Before("pv()")
    public void increPV(){
        ValueOperations valueOperations = redisTemplate.opsForValue();
        Object pv = valueOperations.get("pv");
        if (pv == null){
            valueOperations.set("pv", 1);
        }else {
            Long pv1 = valueOperations.increment("pv");
        }
    }
}
```

在操作鉴权方面（限定只有 `admin` 用户可以访问后台页面的接口），这里我使用了 `Spring` 中的拦截器，来对所有的 `admin` 才能访问的接口进行拦截，判断其携带的 `jwt` 是否合法且属性为 `admin` ，不合法就拦截接口，不让放行。

```
package com.just.controller.interceptor;

import com.just.controller.Result;
import com.just.utils.AuthorityUtil;
import org.springframework.http.HttpMethod;
import org.springframework.web.servlet.HandlerInterceptor;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class AdminInterceptor implements HandlerInterceptor {
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler) throws Exception {
        System.out.println("=====AdminInterceptor=====");
        //放行跨域请求的第一次option请求，解决第二次请求拦截器无法得到header的问题
        if (HttpMethod.OPTIONS.toString().equals(request.getMethod())){
            System.out.println("OPTIONS请求，放行...");
            return true;
```

```
        }
        String uri = request.getRequestURI();
        System.out.println("token = " + request.getHeader("Authorization"));
        System.out.println("token = " + request.getHeader("authorization"));
        boolean isAdmin = AuthorityUtil.isAdmin(request);
        if (!isAdmin){
            System.out.println("defend " + uri);
        }else {
            System.out.println("welcome admin to " + uri);
        }
        return isAdmin;
    }
}
```

在请求的返回响应中，我使用了统一封装结果类来统一返回数据的结果，里面有三个字段，`msg` 存放结果的提示消息，`data` 存储结果的内容，`error` 存放这个结果是否出现了错误，出现了错误就为 `error` ，反之则为空。

```
package com.just.controller;

import com.fasterxml.jackson.annotation.JsonView;
import com.just.utils.AnnotationUtil;
import com.just.utils.JwtUtil;
import lombok.Data;
import org.apache.ibatis.javassist.bytecode.annotation.Annotation;

import java.lang.reflect.Field;

@Data
public class Result {
    private String msg;
    @JsonView
    private Object data;
    private String error;

    public Result(String msg, Object data, String error) {
        this.msg = msg;
        this.data = data;
        this.error = error;
    }

    /**
     * 动态修改@JsonView注解的内容
     * 解决@JsonView不能返回嵌套对象的问题
     * @param msg
     * @param data
```

```
     * @param error
     * @param view
     * @return
     * @throws NoSuchFieldException
     */
    public static Result getResult(String msg, Object data, String error, Class
view) throws NoSuchFieldException {
        Result result = new Result(msg,data, error);
        Field field = result.getClass().getDeclaredField("data");
        AnnotationUtil.modifyAnnoByReflex_field(field, view);
        return result;
    }
}
```

在控制返回数据的结果时，由于数据库的用户表存放了用户的所有消息，但是有时候前端并不需要用户的所有信息，这里我用 `SpringBoot` 提供的 `@JsonView` 注解来动态控制 `controller` 返回的实体类里面的数据有哪些字段，保证安全。

```
package com.just.pojo;


import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonView;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.data.annotation.ReadOnlyProperty;

import java.util.Date;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {
    @JsonView(SessionUser.class)
    private int id;
    @JsonView(SessionUser.class)
    private String username;
    private String password;
    @JsonView(SessionUser.class)
    private String type;
    private String phone;
    private String location;
    private String ps;
    private String intro;
    private String email;
```

```
    private int age;
    private char sex;
    @JsonView(SessionUser.class)
    private String avatar;
    private Date lastLogin;
    private Date registerTime;
    @JsonIgnore
    private boolean del;

    public interface SessionUser{

    }
}
```

```
package com.just.controller;

import com.fasterxml.jackson.annotation.JsonView;
import com.github.pagehelper.PageHelper;
import com.github.pagehelper.PageInfo;
import com.just.service.UserService;
import com.just.service.impl.UserServiceImpl;
import com.just.utils.AuthorityUtil;
import com.just.utils.JwtUtil;
import org.apache.ibatis.annotations.Delete;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import com.just.pojo.User;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import java.util.List;

@RestController
@CrossOrigin
@RequestMapping("/user")
public class UserController {
    @Autowired
    private UserService userService;

    @GetMapping("/add")
    public Result addUser(@RequestBody User user){
        if (!userService.checkUsername(user.getUsername())){
            return new Result("账号已存在", null, "error");
        } else {
            userService.insert(user);
            return new Result(null, null, null);
```

```
        }
    }

    @GetMapping("/all/{page}")
    public Result selectAll(@PathVariable Integer page, HttpServletRequest request)
{
        if (!AuthorityUtil.isAdmin(request)){
            return new Result("无此权限", null, "error");
        }
        PageHelper.startPage(page, 10);
        List<User> users;
        try {
            users = userService.selectAll();
        }catch (Exception e){
            e.printStackTrace();
            return new Result("服务端出现异常", null, "error");
        }
        PageInfo<User> userPageInfo = new PageInfo<>(users);
        List<User> list = userPageInfo.getList();
        return new Result(null, list, null);
    }

    @GetMapping("/size")
    public Result selectAllSize(HttpServletRequest request){
        if (!AuthorityUtil.isAdmin(request)){
            return new Result("无此权限", null, "error");
        }
        return new Result(null, userService.selectAll().size(), null);
    }

    @GetMapping("/{id}")
    public Result selectById(@PathVariable Integer id){
        User user;
        try {
            user = userService.selectById(id);
        }catch (Exception e){
            e.printStackTrace();
            return new Result("服务端出现异常", null, "error");
        }
        return new Result(null, user, null);
    }

    @DeleteMapping("/{id}")
    public Result deleteById(@PathVariable Integer id, HttpServletRequest request){
        if (!AuthorityUtil.isAdmin(request)){
            return new Result("无此权限", null, "error");
```

```
        }
        try {
            userService.deleteById(id);
        }catch (Exception e){
            e.printStackTrace();
            return new Result("服务端出现异常", null, "error");
        }
        return new Result("删除成功", null, null);
    }

    @PostMapping("/updateInfo")
    public Result updateInfo(@RequestBody User user, HttpServletRequest request){
        //检查请求修改的用户id是否和当前用户的id一致,防止窜改请求包的id修改任意用户的信息,否则返回无权限
        if (!AuthorityUtil.isSelf(request, user.getId())){
            return new Result("无此权限", null, "error");
        }
        try {
            userService.updateInfo(user);
        }catch (Exception e){
            e.printStackTrace();
            return new Result("服务端出现异常", null, "error");
        }
        return new Result("更新成功", null, null);
    }

    @PostMapping("/updateType")
    public Result updateType(@RequestBody User user, HttpServletRequest request){
        if (!AuthorityUtil.isAdmin(request)){
            return new Result("无此权限", null, "error");
        }
        userService.updateType(user.getId(), user.getType());
        return new Result(null, null, null);
    }

}
```

在前端需要分页这方面，我使用了 `PageHelper` 这个工具来帮助简化分页的操作。

```
package com.just.controller;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonView;
import com.fasterxml.jackson.core.JsonParser;
import com.github.pagehelper.PageHelper;
import com.github.pagehelper.PageInfo;
```

```java
import com.just.pojo.About;
import com.just.pojo.Passage;
import com.just.pojo.User;
import com.just.service.PassageService;
import com.just.service.UserService;
import com.just.utils.AuthorityUtil;
import com.just.utils.JwtUtil;
import io.jsonwebtoken.Jwt;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.util.Date;
import java.util.List;

@RestController
@CrossOrigin
@RequestMapping("/passage")
public class PassageController {
    @Autowired
    private PassageService passageService;

    @Autowired
    private UserService userService;

    /**
     * 分页返回所有的文章
     * admin可以访问
     * @param page
     * @return
     */
    @GetMapping("/all/{page}")
    public Result selectAllPassage(@PathVariable Integer page, HttpServletRequest
request) {
        if (!AuthorityUtil.isAdmin(request)) {
            return new Result("无此权限", null, "error");
        }
        PageHelper.startPage(page, 10);
        List<Passage> passageList = passageService.selectAllPassage();
        if (passageList == null || passageList.isEmpty()) {
            return new Result("暂无数据", null, "error");
        } else {
            PageInfo<Passage> passagePageInfo = new PageInfo<>(passageList);
            List<Passage> list = passagePageInfo.getList();
```

```
            return new Result(null, list, null);
        }
    }

    /**
     * 处理首页的文章列表
     *
     * @param page
     * @return
     * @throws NoSuchFieldException
     */
    @JsonView(Passage.IndexPassage.class)
    @GetMapping("/index/{page}")
    public Result selectIndexPassage(@PathVariable Integer page, HttpServletRequest
request) throws NoSuchFieldException {
        PageHelper.startPage(page, 5);
        List<Passage> passageList = passageService.selectIndexPassage();
        if (passageList == null || passageList.isEmpty()) {
            return new Result("暂无数据", null, "error");
        } else {
////            return  new Result(null, passageList, null);
            PageInfo<Passage> passagePageInfo = new PageInfo<>(passageList);
            List<Passage> list = passagePageInfo.getList();
            return Result.getResult(null, list, null, Passage.IndexPassage.class);
//            return new Result(null, list, null);
        }
    }

    /**
     * 根据文章id获取文章详情
     *
     * @param passageId
     * @return
     */
    @GetMapping("/{passageId}")
    public Result selectOneByPassageId(@PathVariable Integer passageId) {
        Passage passage = passageService.selectById(passageId);
        return new Result(null, passage, null);
    }

    /**
     * 获取所有文章的总数
     *
     * @return
     * @throws NoSuchFieldException
     */
```

```
    @GetMapping("/size")
    public Result selectAllPassageSize(HttpServletRequest request) throws
NoSuchFieldException {
        //startPage要在sql查询的前面加才能生效
        PageHelper.startPage(1, 5);
        List<Passage> passageList = passageService.selectIndexPassage();
        PageInfo<Passage> passagePageInfo = new PageInfo<>(passageList);
        long size = passagePageInfo.getTotal();
        return new Result(null, size, null);
    }

    /**
     * 获取当前登录用户的文章总数
     * @param request
     * @return
     * @throws NoSuchFieldException
     */
    @GetMapping("/my/size")
    public Result selectMyPassageSize(HttpServletRequest request) throws
NoSuchFieldException {
        //startPage要在sql查询的前面加才能生效
        PageHelper.startPage(1, 5);
        int id = getIdFromRequest(request);
        List<Passage> passageList = passageService.selectPassageByAuthorId(id);
        PageInfo<Passage> passagePageInfo = new PageInfo<>(passageList);
        long size = passagePageInfo.getTotal();
        return new Result(null, size, null);
    }

    /**
     * 分页获取某个作者的所有文章
     *
     * @param page
     * @param size
     * @param request
     * @return
     * @throws NoSuchFieldException
     */
    @GetMapping("/my/{page}/{size}")
    public Result selectByAuthorId(@PathVariable Integer page, @PathVariable
Integer size, HttpServletRequest request) throws NoSuchFieldException {
        int id = getIdFromRequest(request);
        if (id == -1) {
            return new Result("请先登录", null, "error");
        }
```

```
        PageHelper.startPage(page, size);
        List<Passage> passageList = passageService.selectPassageByAuthorId(id);
        PageInfo<Passage> passagePageInfo = new PageInfo<>(passageList);
        List<Passage> list = passagePageInfo.getList();

        if (list == null || list.isEmpty()) {
            return new Result("empty", null, "error");
        }
        return Result.getResult(null, list, null, Passage.TablePassage.class);
    }

    /**
     * 删除某个文章
     *
     * @param passageId
     * @param request
     * @return
     */
    @DeleteMapping("/{passageId}")
    public Result deleteById(@PathVariable Integer passageId, HttpServletRequest
request) {
        int userId = getIdFromRequest(request);
        Passage passage = passageService.selectById(passageId);
        //判断前端要删除的文章是否存在
        if (passage == null) {
            return new Result("所要删除的文章不存在", null, "error");
        }
        //判断要求删除的用户是不是admin
        if (AuthorityUtil.isAdmin(request)) {
            passageService.deletePassage(passageId);
            return new Result(null, null, null);
        } else if (passage.getAuthorId() == userId) {
            //判断要求删除的用户是不是该文章的作者
            passageService.deletePassage(passageId);
            return new Result(null, null, null);
        } else {
            return new Result("无此权限", null, "error");
        }
    }

    /**
     * 更新某个文章的内容
     *
     * @param passage
     * @param request
     * @return
```

```
     */
    @PostMapping("/")
    public Result update(@RequestBody Passage passage, HttpServletRequest request)
{
        int userId = getIdFromRequest(request);
        int passageId = passage.getId();
        Passage realPassage = passageService.selectById(passageId);
        //用户越权更新别人的文章
        if (realPassage.getAuthorId() != userId) {
            System.out.println(realPassage.getAuthorId());
            return new Result("无此权限", null, "error");
        } else {
            passageService.updatePassage(passage);
            return new Result(null, null, null);
        }
    }

    /**
     * 添加文章
     *
     * @param passage
     * @param request
     * @return
     */
    @PostMapping("/add")
    public Result add(@RequestBody Passage passage, HttpServletRequest request) {
        //从token中获取当前发布文章的作者
        int authorId = getIdFromRequest(request);
        int id = passageService.add(passage, authorId);

        return new Result(null, id, null);
    }

    @PostMapping("/like/{passageId}")
    public Result increaseLikeNum(@PathVariable Integer passageId) {
        passageService.increaseLikeNum(passageId);
        return new Result(null, null, null);
    }

    @GetMapping("/about")
    public Result getAbout() {
        return new Result(null, passageService.selectAbout(), null);
    }

    @PostMapping("/about")
    public Result updateAbout(@RequestBody About about, HttpServletRequest request)
```

```
{
        if (!AuthorityUtil.isAdmin(request)) {
            return new Result("无此权限", null, "error");
        }
        passageService.updateAbout(about.getContent());
        return new Result(null, null, null);
    }

    /**
     * 从request获取用户id的工具方法，简化controller层的代码
     *
     * @param request
     * @return
     */
    private int getIdFromRequest(HttpServletRequest request) {
        try {
            String token = request.getHeader("Authorization");
            String username = JwtUtil.getUsernameByToken(token);
            User user = userService.selectByUsername(username);
            return user.getId();
        } catch (Exception e) {
            return -1;
        }
    }
}
```

参考资料

`SpringBoot` 官网，`Vue` 官方手册，`elementUI` 官方手册，`CSDN` .....