目录

第一章 绪论

1.1 系统的目的和意义

本系统定位为一个简易的博客系统，最主要功能为博客的发布、浏览、收藏、搜索、用户管理等。随着计算机技术的发展和昔及，人们对资源共享的需求也在不断地增强。Blog 是 Weblog 的简称，它是继 E-mail、BBS、ICQ 之后出现的第 4 种新的生活方式、新的工作方式、新的 学习方式和新的网络交流方式◊ Blog 主要应用于 3 个方面：一是新的人际交流通道；二是 以个人为中心的信息摘选和知识管理平台；三是以个人为中心的传播出版资源库。blog 作为个人的一种学习工具，简单易用，广泛吸引着人们的兴趣。Blog 现在在教育方面、商业 方面、公司内部、校园领域等都得到了很大的发展，它将互联网从过去的通讯功能、资料功能、交流功能等进一步强化，使其更加个性化、开放化、实时化、全球化，把信息共享发展 到资源共享、思想共享、生命历程共享。Blog 已经成为一种继课件、积件、资源库、教育 主题网站等信息化教学模式之后，新的网络应用模式。它是应时代的需求而兴起的，也是应时代需求不断进步的。现在博客是人们学习和交流的主要方法之一，得到社会的广泛欢迎和需求。所以我在本课题中选择做一个博客系统，希望能给大家交流提供一个简易、方便的平台。

第二章 博客系统需求分析

2.1 引言

本系统定位为一个简易博客系统，实现了博客浏览、发布、收藏、搜索、用户管理(包括用户的登录、退出、注册、注销、基本信息修改等)。
系统开发过程中前后端完全分离。前端采用 MVVM 设计模式，使用 Vue 作为视图层框架，Vuex 实现状态管理，使用 element UI 组件库，使用 axios 库进行数据的异步请求。后端使用语言使用 Java，采用 MVC 设计模式，接口设计时遵守常用接口设计规范(如：单一职责原则、接口隔离原则)

2.2 需求分析

2.2.1 用户管理需求分析

1.用户注册：

- 为初次使用本系统的用户提供注册功能
- 初始化用户信息(包括用户头像、性别、简介等)

2.用户登录与退出：

- 如果想要使用系统全部功能，请先登录
- 用户登录过程中需要使用到验证码
- 用户一次登录后记住用户信息，不必每次重复登录
- 退出登录后清除用户登录信息，下次使用系统需重新登录

3.用户权限管理：

- 未登录用户能使用的功能只包括：博客浏览和用户注册
- 登录用户能够使用的功能模块包括：博客发布、博客收藏、用户信息修改、账号注销、查看自己发布的文章、查看自己收藏的文章

4.用户信息修改：

- 用户可以修改自己的基本信息，包括：头像上传、昵称、电话号码、简介、性别等基本细信息

5.用户注销

- 如果不想再使用本系统，提供用户注销的功能
- 用户注销后会将用户信息全部删除，无法找回

2.2.2 博客管理需求分析

1.查看已发布博客

- 所有使用本系统的用户都可以浏览本系统所有发布的文章
- 提供查询功能，方便用户针对性阅读

2.发布博客

- 登录用户可以发布博客，分享自己的看法与观点
- 博客发布功能使用富文本编辑器，方便用户编辑属于自己风格、简洁优美的文章

3.博客收藏

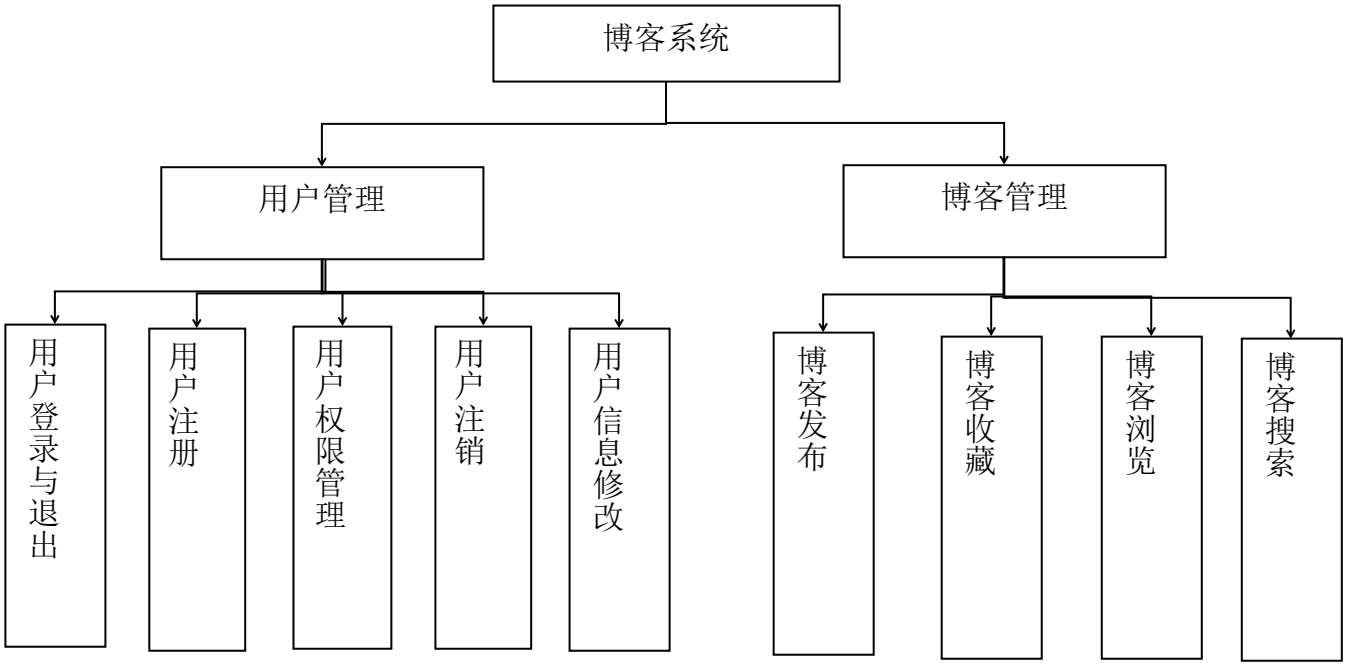
- 已登录用户可以收藏自己喜欢的文章，方便多次查看
- 如果不再需要已收藏的文章，可取消收藏
- 记录每篇博客的收藏人数，方便用户寻找优质文章

第三章 博客系统结构分析与设计

3.1 引言

以下进行系统分析与设计，分析目标为：识别用户需求、评价系统可行性、系统模块化设计。

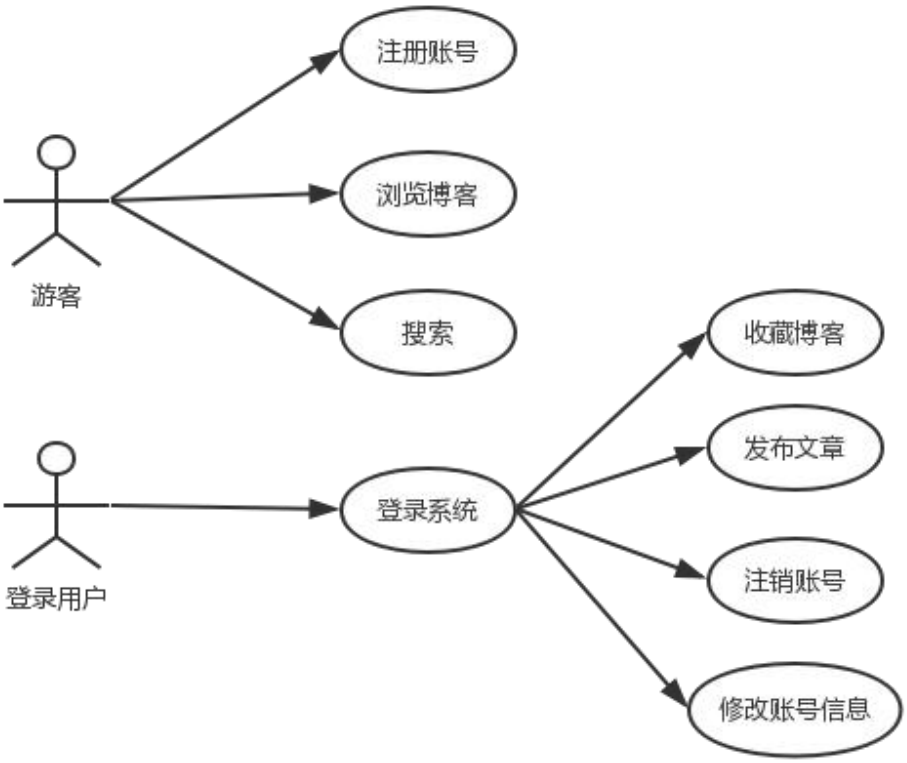
3.2 系统模块结构图



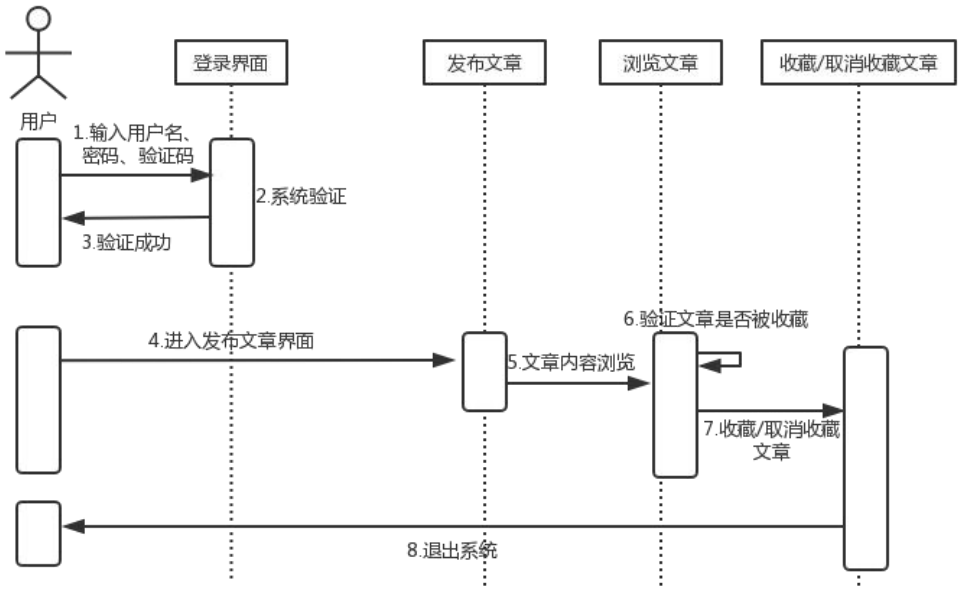
3.3 用户管理模块

1.概述

本系统用户身份分为两种：游客和登录用户。游客只能使用系统部分功能：注册、博客浏览和搜索，登录用户可以：发布文章、收藏文章、修改账号信息和注销账号。


注册账号
浏览博客
游客
搜索
收藏博客
发布文章
登录系统
注销账号
登录用户
修改账号信息

3.4 博客管理模块


登录界面
发布文章
浏览文章
收藏/取消收藏文章
用户
1.输入用户名、
密码、验证码
2.系统验证
3.验证成功
4.进入发布文章界面
5.文章内容浏览
6.验证文章是否被收藏
7.收藏/取消收藏
文章
8.退出系统

第四章 关键技术研究

4.1 实现技术路线

1.数据库使用 mysql 数据库

MySQL 是一个小型关系型数据库管理系统，支持 FreeBSD、Linux、MAC、Windows 等多种操作系统，虽然与其他的大型数据库例如 Oracle、DB2、SQL Server 等相比功能稍弱一些，但是它的优点如下：

- 可以处理拥有上千万条记录的大型数据；
- 支持常见的 SQL 语句规范；
- 可移植行高，安装简单小巧；
- 良好的运行效率，有丰富信息的网络支持；
- 调试、管理，优化简单（相对其他大型数据库）

2.客户端使用 Vue 框架

- 客户端使用 Vue 作为前端框架

  Vue 是一款轻巧、高性能、可组件化的前端框架，同时拥有非常容易上手的 API，能缩短开发周期，提高开发效率，同时使产品拥有更高的性能。

- 使用 Vuex 进行状态管理

  Vue 定位只是有个视图层框架，本系统中使用 Vuex 作为数据层框架，更高效地进行数据处理。

- 组件库使用 Element UI

  Element 是一款很简单、漂亮的前端开发库，能与 Vue 完美融合，使用简单方便，本系统将其作为前端组件库，能够快速的搭建简洁美观的前端页面。

- 异步数据请求使用 axios 库

  axios 是一个基于 promise 的 HTTP 库，支持 promise 的所有 API，它可以拦截请求和响应，并且可以转换请求数据和响应数据，并对响应回来的内容自动转换为 json 类型的数据，安全性更高，客户端支持防御 XSRF。

3.服务端使用 JAVA 语言

目前流行的后端开发语言有 PHP、ASP.net，JAVA 等，JAVA 对比其他开发语言虽然开发速度比较慢，但是 JAVA 的框架越来越完善，具有卓越的通用性、高效性、

平台移植性和安全性。

4.开发方式采用前后端分离

开发过程为:需求分析——约定接口——前后端并行开发——前后端集成——前端页面调整——集成成功——交付。才用前后端分离的优势为:

- 实现真正的前后端解耦，动态资源和静态资源分离，提高了性能和扩展性
- 前端静态化

  前端有且仅有静态内容，再明确些，只有 HTML/CSS/JS；其内容来自于完全静态的资源而不需要任何后台技术进行动态化组装；前端内容的运行环境和引擎完全基于浏览器本身。

- 后端数据化

  后端可以用任何语言，技术和平台实现；遵循一个原则：只提供数据，不提供任何和界面表现有关的内容；统一 API 接口，接口完全可以共用。

- 架构分离化

  前端架构完全基于 HTML/CSS/JS 的发展和 js 框架的演变，由于前台是纯静态内容，大型架构方面可以考虑向 CDN 方向发展。后端架构几乎可以基于任何语言和平台的任何解决方案，大型架构方面，RESTful Api 可以考虑负载均衡，而数据、业务实现等可以考虑数据库优化和分布式。在大并发情况下，可以同时水平扩展前后端服务器。

- 减少后端服务器的并发压力，除了接口以外的其他所有 http 请求全部转移到前端服务器上，页面不再是全部刷新，而是异步加载，局部刷新，减轻压力。
- 安全性方面的集中优化

  前端静态以后，一些注入式攻击在分离模式下被很好的规避，而后端安全问题集中化了，主要考虑处理 RESTful 接口安全，安全架设和集中优化变得更明确和便利

4.2 关键技术研究

4.2.1 数据库设计

1.Users——记录用户登录信息

字段名	类型	长度	不是 null	键	注释
Id	Int	8	√	主键	用户唯一标识
username	varchar	30	√	--	用户名
password	varchar	30	√	--	用户密码

2. Userinfo——记录用户个人信息

字段名	类型	长度	不是 null	键	注释
Id	Int	8	√	主键	用户唯一标识
avatar	varchar	30	√	--	头像
Introduction	Text	--		--	自我介绍
Collection	Varchar	255		--	收藏文章
Sex	Bit	--		--	性别
Tel	varchar	11		--	电话号码

3. Article——记录发表文章

字段名	类型	长度	不是 null	键	注释
Id	Int	8	√	主键	文章唯一标识
Feature	varchar	30	√	--	文章特色图标
AuthorId	int	8	√	--	文章作者 ID

Title	Varchar	255	√	--	文章标题
Content	Text	--	√	--	文章内容
Created	Datetime	--	√	--	发布时间
Likes	Int	8	√	--	喜欢人数

4.2.2Vue 构建前端页面

- 本系统为一个单页面应用程序，使用 vue-router 实现前端路由：

```
import Router from 'vue-router'
Vue.use(Router)
export default new Router({
    ......
})
```

- 系统使用 Vuex 进行状态管理：

```
import Vue from 'vue'
import Vuex from 'vuex'
import state from './state'
import mutations from './mutations'
import actions from './actions'
import getters from './getters'

Vue.use(Vuex)
export default new Vuex.Store({
  state,
  getters,
  actions,
  mutations,
})
```

- 使用 Element UI 组件库

```
import ElementUI from 'element-ui'
import 'element-ui/lib/theme-chalk/index.css'
Vue.use(ElementUI)
```

当全局注册 Element UI 后(如上)，可以直接以标签的形式使用 Element UI 组件库提供的组件，如使用一个输入框：

```
<el-input
  placeholder="请输入内容"
  prefix-icon="el-icon-search"
  class="search-input"
  v-model="searchInfo"
  @keyup.enter.native="search">
</el-input>
```

- 使用 axios 进行数据的异步请求

```
import axios from 'axios'
Vue.prototype.$axios = axios
```

值得注意的一点是，axios，默认使用的数据格式为 json，服务端使用表单的接收方式会无法获取，需要进行全局配置：

```
axios.defaults.headers.post['Content-Type'] =
'application/x-www-form-urlencoded';
axios.defaults.headers.get['Content-Type'] =
'application/x-www-form-urlencoded';
axios.defaults.transformRequest = [function (data) {
  let ret = ''
  for (let it in data) {
    ret += encodeURIComponent(it) + '=' +
encodeURIComponent(data[it]) + '&'
  }
  return ret
}]
```

4.2.3 前端路由守卫

本系统是一个单页应用程序，使用组件路由的方式。前端为了过滤掉跳过登录模块直接访问组件的非正常访问，使用路由守卫的方式。使用 router.beforeEach 注册一个全局前置守卫：

```
const router = new VueRouter({ ... })
router.beforeEach((to, from, next) => {
    // ...
})
```

- to: Route: 即将要进入的目标 路由对象。
- from: Route: 当前导航正要离开的路由。
- next: Function: 下一步操作。当用户是未登录状态，并且要访问登录用户才能访问的组件时，使用钩子函数 next(‘/login’)跳转到登录页面。

4.2.4 数据接口设计

1. 用户登录

接口名称：/api/Login				
请求方法：Post				
请求参数	属性	描述	是否必填	类型
	username	用户名	是	String
	password	用户密码	是	String
返回参数	属性	描述	是否必填	类型
	--	成功返回登录用户信息，包括用户名、密码和用户 id；失败返回空	是	Json 对象

2. 用户注册

接口名称：/api/Register				
请求方法：Post				
请求参数	属性	描述	是否必填	类型
	username	用户名	是	String
	password	用户密码	是	String
返回参数	属性	描述	是否必填	类型
	success	返回：true 表示注册成功；返回：false 表示注册失败	是	String

3. 获取文章列表

接口名称：/api/getArticle				
请求方法：get				
请求参数	属性	描述	是否必填	类型
	page	获取第几页内容	是	Int
	isHome	是否第一次获取(即是否需要返回总页数)	否	Boolean
返回参数	属性	描述	是否必填	类型
	totalpage	返回文章总页数	否	Int
	articles	返回文章列表	是	Array

4.获取文章具体内容

接口名称：/api/getContent
请求方法：get

请求参数	属性	描述	是否必填	类型
	id	文章 ID	是	Int
返回参数	属性	描述	是否必填	类型
	title	文章标题	是	String
	content	文章内容	是	String
	author	作者	是	String
	created	发布时间	是	Datetime
	likes	收藏人数	是	Int

5.获取当前用户信息

接口名称：/api/getUserInfo				
请求方法：get				
请求参数	属性	描述	是否必填	类型
	id	用户 id	是	Int
返回参数	属性	描述	是否必填	类型
	avatar	用户头像	是	Int
	sex	用户性别	是	Bit
	introduction	用户简介	是	String
	tel	用户电话号码	是	String

6.获取当前用户所写文章

接口名称：/api/getUserArticle

请求方法：get				
请求参数	属性	描述	是否必填	类型
	userId	用户 id	是	Int
返回参数	属性	描述	是否必填	类型
	--	文章列表	是	Array

7.获取当前用户收藏文章

接口名称：/api/getCollection				
请求方法：get				
请求参数	属性	描述	是否必填	类型
	userId	用户 id	是	Int
返回参数	属性	描述	是否必填	类型
	--	文章列表	是	Array

8.修改当前用户信息

接口名称：/api/changeUserInfo				
请求方法：post				
请求参数	属性	描述	是否必填	类型
	id	用户 id	是	Int
	username	用户昵称	是	String
	sex	用户性别	是	Bit
	tel	用户电话	是	String
	introduction	用户简介	是	String
返回参数	属性	描述	是否必填	类型
	--	返回 true 表示修改成功；	是	Boolean

		返回 false 表示修改失败		

9.头像上传

接口名称：/api/changeAvatar				
请求方法：post				
请求参数	属性	描述	是否必填	类型
	id	用户 id	是	Int
	image	用户头像	是	File
返回参数	属性	描述	是否必填	类型
	--	返回 true 表示上传成功； 返回 false 表示上传失败	是	Boolean

10.判断当前用户是否收藏否一篇文章

接口名称：/api/judgeCollection				
请求方法：get				
请求参数	属性	描述	是否必填	类型
	userId	用户 id	是	Int
	articleId	文章 id	是	Int
返回参数	属性	描述	是否必填	类型
	--	返回 true 表示文章已收藏； 返回 false 表示文章未收藏	是	Boolean

11.收藏文章/取消收藏

接口名称：/api/judgeCollection				
请求方法：get				
请求参数	属性	描述	是否必填	类型
	userId	用户 id	是	Int
	articleId	文章 id	是	Int
	how	具体操作：add 表示收藏文章，delete 表示取消收藏	是	String
返回参数	属性	描述	是否必填	类型
	--	返回 true 表示文章成功； 返回 false 表示失败	是	Boolean

12.用户注销

接口名称：/api/deleteUser				
请求方法：get				
请求参数	属性	描述	是否必填	类型
	userId	用户 id	是	Int
返回参数	属性	描述	是否必填	类型
	--	返回 true 表示注销成功； 返回 false 表示注销失败	是	Boolean

13.文章搜索

接口名称：/api/search

请求方法：get				
请求参数	属性	描述	是否必填	类型
	searchInfo	关键词	是	String
返回参数	属性	描述	是否必填	类型
	--	返回文章列表	是	Array

14.发布文章

接口名称：/api/addArticle				
请求方法：Post				
请求参数	属性	描述	是否必填	类型
	id	用户 id	是	Int
	title	文章标题	是	String
	content	文章内容	是	String
返回参数	属性	描述	是否必填	类型
	success	返回：true 表示发布成功；返回：false 表示发布失败	是	String

4.2.5 数据库操作封装

将数据库的操作单独封装为一个类 DBUtiles，包含曾、删、改、查操作和关闭数据库连接。查询操作返回的数据格式为 JSON。

1.数据库连接

```
……
//数据库链接
public DBUtils() {
    className="com.mysql.jdbc.Driver";
```

```
url="jdbc:mysql://localhost:3306/blog?useSSL=false&characterEncoding=utf8";
    username="fusiyu";
    password="111111";
    conn = null;
    statement = null;
    try{
        Class.forName(className);
        conn = DriverManager.getConnection(url, username, password);
    }catch(Exception e){
        System.out.println("加载数据库驱动程序失败！");
        e.printStackTrace();
    }
}
```

2. 封装查询操作，public ResultSet doSelect(String sql)，返回的结果类型为 ResultSet，传入 SQL 语句。

3. 封装将 ResultSet 数据类型转化为，JSON 字符串的函数。要是用 JAON 对象，需要下载 json-org.jar 包，关键代码如下：

```
// json 数组
JSONArray array = new JSONArray();
……
// 遍历 ResultSet 中的每条数据
 while (rs.next()) {
     JSONObject jsonObj = new JSONObject();

     // 遍历每一列
     for (int i = 1; i <= columnCount; i++) {
         String columnName =metaData.getColumnLabel(i);
         String value = rs.getString(columnName);
         jsonObj.put(columnName, value);
     }
```

```
    array.put(jsonObj);
 }
……
```

4. 封装增删改操作的函数 public Boolean excute(String sql)，传入 SQL 语句，返回布尔值。

5. 封装关闭连接的函数，public void closeConnection().

4.2.6session+fileter

服务端使用 session 技术管理在线用户，使用 filter 过滤掉跳过登录系统直接访问的非正常访问。

- 用户登录成功后，使用 Session 存储当前用户用户名.

  request.getSession().setAttribute(“cur_user”,username)

- 用户退出登录后，删除对应 Sesion。

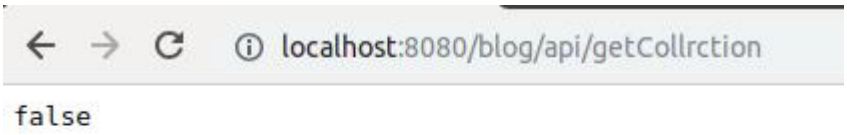
  request.getSession().removeAttribute

- 编写对应 filter

```
//强制转换
HttpServletRequest req = (HttpServletRequest) request;
HttpServletResponse res = (HttpServletResponse) response;

//获取资源路径
String uri = req.getRequestURI();

//访问未登录用户可以访问的资源
if(uri.contains("login")||uri.contains("getArticle")||uri.con
tains("getContent")||uri.contains("register")||uri.contains("
search")) {
   chain.doFilter(request, response);
}else {
   String cur_user = (String)
```

```
req.getSession().getAttribute("cur_user");
   System.out.println("filter"+cur_user);
   if(cur_user != null) {
      chain.doFilter(request, response);
   }else {
      PrintWriter out = res.getWriter();
      out.print("false");
   }
}
```

- 测试效果，当未登录用户直接访问接口，获取登录登录收藏的文章，如下：



4.2.7 文件上传

在系统中头像上传的过程中需要用到文件上传的技术，具体实现中需要用到：commons-fileupload-1.3.2.jar 和 commons-io-2.5.jar，关键代码如下：

```
try {
  //解析即被包装之后的 HttpServletRequest 对象，既是分离文本表单和上传文件（http 请求会被包装为 HttpServletRequest）
  List<FileItem> items = upload.parseRequest(request);
  for(FileItem item:items){
    String fieldName = item.getFieldName();
    String fileName = item.getName();
    String contentType = item.getContentType();
    boolean isInMemory = item.isInMemory();
    long sizeInBytes = item.getSize();
    //实例化一个文件
```

```
    //request.getRealPath(获取真实路径)
    String Image_Path = getServletContext().getRealPath("/") +
File.separator + UPLOAD_DIRECTORY;
    Image_name =
fileName.substring(fileName.lastIndexOf("\\")+1,fileName.leng
th());
    File file = new File(Image_Path+Image_name);
    item.write(file);
  }
```

第五章 系统实现

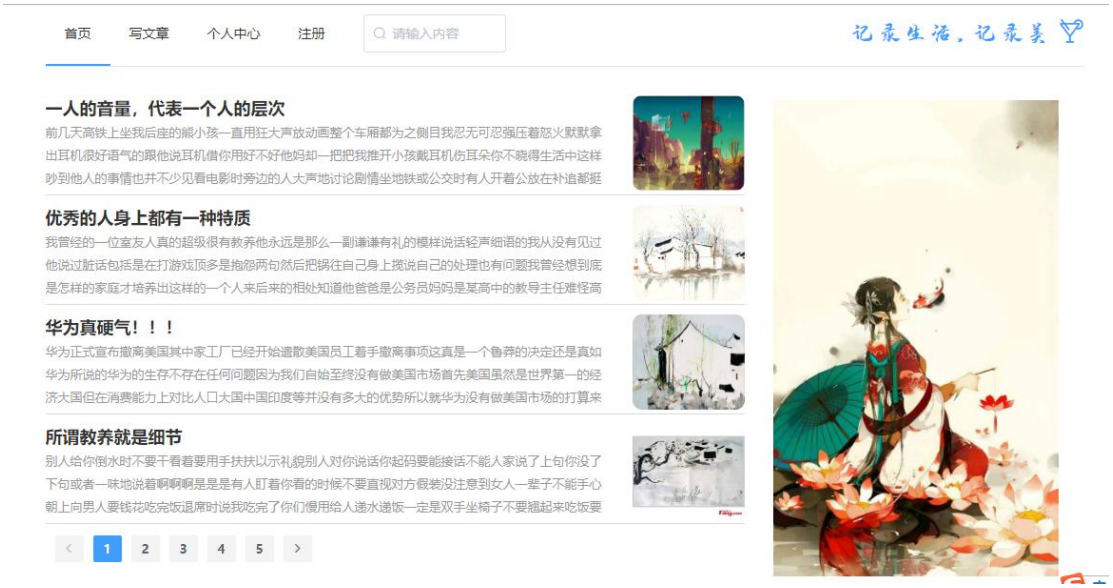
5.1 系统实现介绍

5.1.1 博客浏览

- 进入系统首页分页展示发布的博客
  前端实现：“article”为文章列表，数据类型为数组，遍历该数组，展示文章具体内容，代码如下：

```
<div class="list-item" v-for="item in articles" :key="item.id">
  <img
class="pic" :src="`${imgUrl}image/article/feature/${item.featu
re}`"/>
  <div class="article-info">
    <router-link class="title"
v-html="item.title" :to="{path:'/detail',query:{id:item.id}}
"/>
    <p class="desc"
v-text="item.content.match(reg).join('')"></p>
  </div>
</div>
```

效果图如下：



- 点击不同页码展示不同内容

```
<!--分页组件-->
<el-pagination
  v-if="!ifSearch"
  background
  layout="prev, pager, next"
  :page-count="totalPage"
  @current-change="getAriticle">
</el-pagination>
```

效果图如下：

- 关键词搜索

  当用户按下回车键时，获取用户输入关键词，请求相应数据接口，展示返回内容。

```
search(){
this.$axios.get('/api/search?searchInfo='+this.searchInfo).th
en((res) => {
```

```
    this.$store.dispatch('changeArticles', res.data)
    this.$store.dispatch('changeIfSearch', true)
  })
}
```

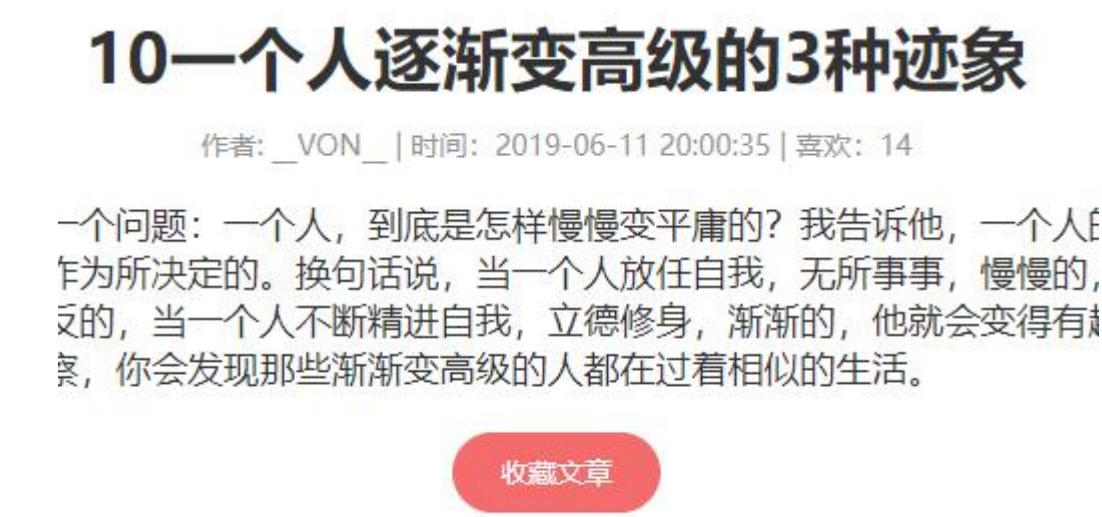
效果图如下：

- 文章具体内容展示

  获取文章 id，请求相应数据接口，获取返回内容，进行文章内容展示。此处使用了前端路由，使用 query 的方式进行路由传值，代码如下：

```
<router-link class="title" v-html="item.title"
:to="{path:'/detail',query:{id:item.id}}" />
......
created() {
  this.id = this.$route.query.id
}
```

效果图如下：



5.1.2 用户登录

- 输入用户名、密码、验证码

  效果图如下：

- 判断用户名和密码是否为空，是则中断操作

  前端实现：当用户按下登录按钮后，获取用输入的用户名和密码，用户名和密码为空，给出错误提示信息，中断操作，代码如下：

```
if(!this.username){
  this.$message({
    showClose: true,
    message: '用户名不能为空',
    type: 'error'
  })
  return
}
if(!this.password){
  this.$message({
    showClose: true,
    message: '密码不能为空',
    type: 'error'
  })
  return
}
```

效果图如下：

- 判断验证码是否正确，否则中断操作

  当用户按下登录按钮后，获取用户输入验证码，判断和随机生成的验证码是否相等，不相同则中断操作，代码如下：

```
if(this.inputIdentify !== this.identify){
  this.$message({
    showClose: true,
    message: '验证码错误',
    type: 'error'
  })
  return
}
```

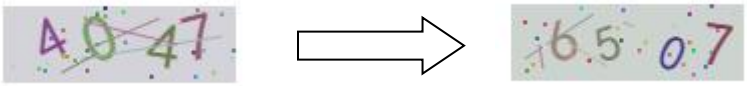
效果图如下：

- 如若验证码不清晰，可以点击验证码进行切换

  前端实现：点击验证码，随机生成四位数的验证码，代码如下：

```
//切换验证码
randomNum() {
  this.identify = (Math.floor(Math.random() * 10000))
.toString()
}
```

效果图如下：



- 调用相应后台接口验证用户身份，返回 true 表示登录成功，返回 false 表示登录失败

  前端实现：根据服务端返回数据，判断用户是否登录成功，是则更新当前用户信息，为了避免每次进入系统都要重新登录，使用 localstorage 存储当前用户信息和登录状态，然后跳转到相应界面；登录失败则给出相应提示，代码如下：

```
{
 ......
  localStorage.setItem("userLogin",true)
 localStorage.setItem("cur_user",JSON.stringify(res.data[0]))
 this.$store.dispatch('userLogin',res.data[0])
 this.$router.push('/userinfo/basic')
}
```

5.1.3 用户注册

- 输入用户名和密码

  效果图如下：

- 用户名和密码不能为空，为空则中断操作，给出提示信息

  效果图如下：

- 确定密码是否输入正确，两次密码不一致，中断操作，给出提示

  效果图如下：

- 请求对应后端接口，注册用户，根据返回信息判断用户是否登录成功

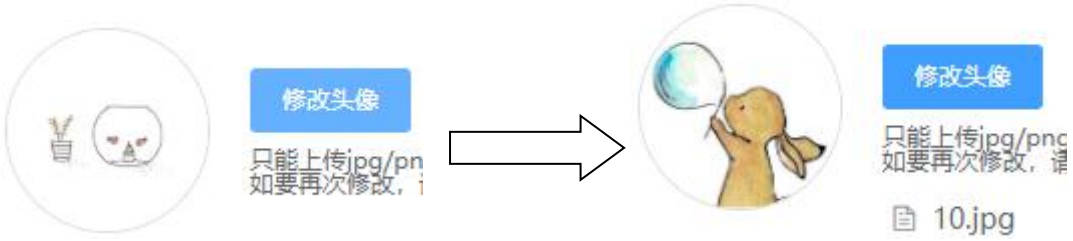
5.1.4 用户信息修改

- 用户头像修改

  代码如下：

```
<img :src="`${imgUrl}image/user/${userinfo.avatar}`"
class="avatar">

<el-upload
  class="upload-demo update-btn"
  :action="`/api/changeAvatar?id=${cur_user.id}`"
  :multiple="false"
  :limit="1"
  accept="image/jpg,image/png"
  :on-change="updateAvatar">
  <el-button size="small" type="primary">修改头像</el-button>

......

updateAvatar(file, fileList) {
    const _this = this
    let reader = new FileReader()
    reader.readAsDataURL(fileList[0].raw)
    reader.onload = function () {
      _this.imgUrl = reader.result

    }
}
```

效果图如下：



- 用户基本信息修改

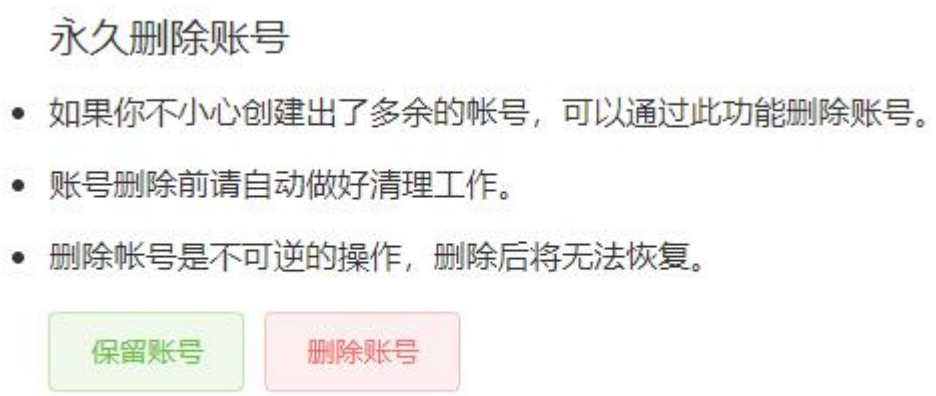
  获取用户修改后的基础信息，调用相应数据接口修改用户信息，根据服务端返回判断是否修改成功。效果图如下：

5.1.5 查看相关文章

- 登录用户可以查看自己发布的文章
- 登录用户可以查看自己收藏的文章，效果图如下：

5.1.6 用户注销

- 当用户点击注销账号的按钮后，获取当前用户的 id，调用相应后端数据接口，删除相应用户，效果图如下：



5.1.7 用户退出

- 当用户点击退出按钮后，清除本地用户信息，删除对应 session，回到博客首页。

5.1.8 发布博客

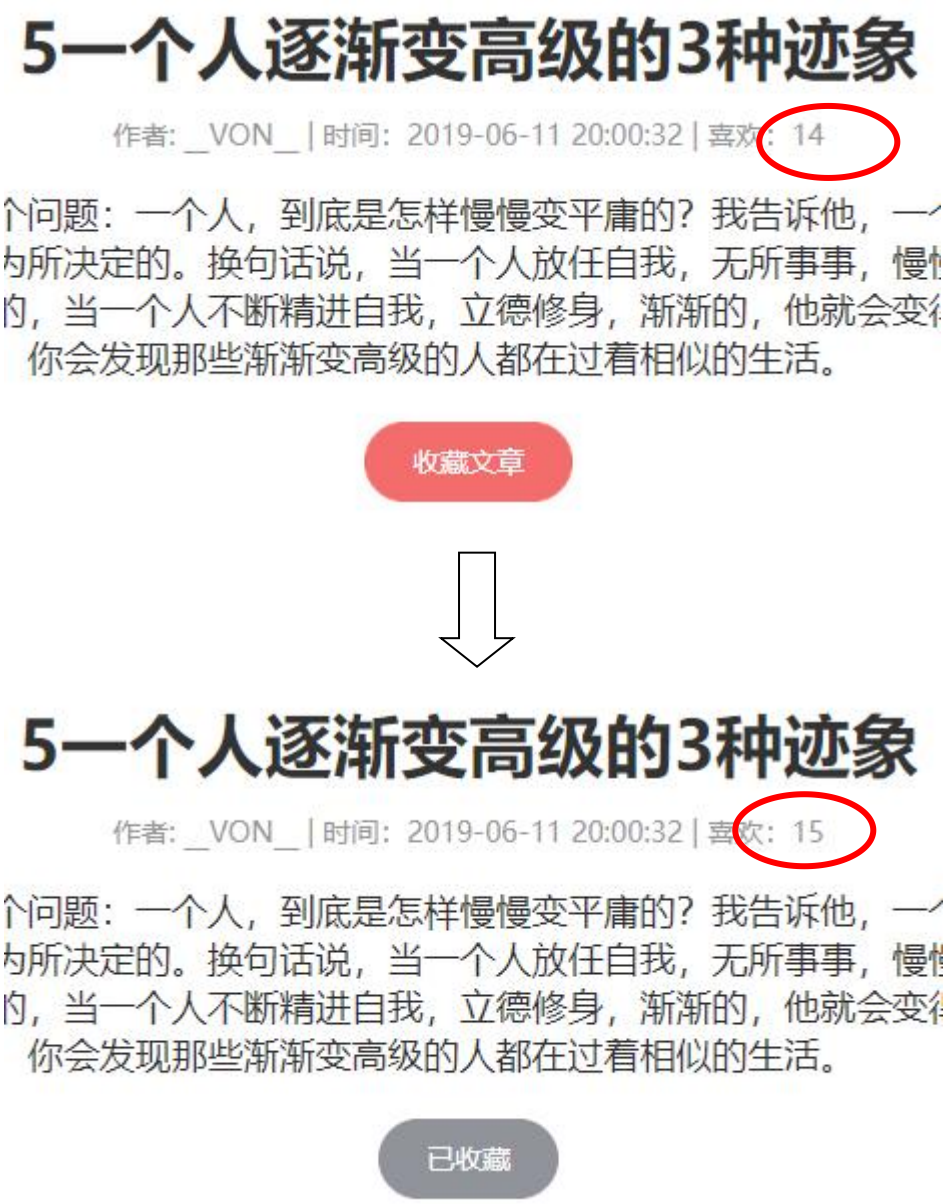
- 只有登录用户可以发布文章，用户未登录点击发布文章按钮会自动跳转到登录页面
- 文章标题内容不能为空
- 当用户点击发布按钮后，获取当前用户 id、用户输入的文章内容和文章标题，调用相关接口，存储本片文章，效果图如下：

5.1.9 博客收藏/取消收藏

- 只有登录用户可以收藏文章，未登录用户点击收藏文章，会自动跳转到登录页面
- 在组件 create 生命周期中，获取当前文章 id 和用户 id，调用相关接口，判断当前用户是否已收藏该文章，是则显示已收藏按钮，否则显示收藏按钮，代码如下：

```
<el-button type="danger" v-if="!isCollection" round
class="collection-btn" @click="collectionArticle">收藏文章
</el-button>
  <el-button type="info" v-else round class="collection-btn"
@click="unCollectionArticle">已收藏</el-button>
……
created() {
let loginState = this.$store.getters.getLoginState
  if(loginState){
    //判断当前用户是否收藏文章
    let userId = this.cur_user.id
    let articleId = this.id
this.$axios.get('/api/judgeCollection?articleId='+articleId+"
&userId="+userId).then((res) => {
      if(res.data){
        this.isCollection = true
      }
    })
  }
}
```

- 点击收藏按钮，获取当前用户 id 和文章 id，并且传递给后端的参数“how”设置为“add”表示收藏文章。服务端收到请求后给对应用户收藏的文章增加该项，文章对应的喜欢人数加一，前端数据同步变化，效果如下：



- 点击已按钮，获取当前用户 id 和文章 id，并且传递给后端的参数“how”设置为“delete”表示取消收藏文章。服务端收到请求后给对应用户收藏的文章删除该项，文章对应的喜欢人数减一，前端数据同步变化。效果类似收藏。

5.2 系统实现的不足

- 没有验证用户名是否重复
- 没有验证用户上传的头像是否为图片
- 上传头像需要手动更新才能才能看到效果(当网速慢的时候，头像会出现较长时间的空白)