MyAritist 图片风格迁移应用
详细设计说明书

拟 制 人_____董建文_____

审 核 人_____

批 准 人_____

[二零一九年四月二十日]

详细设计说明书

1.引言

1.1 编写目的

本阶段完成系统的大致设计并明确系统的数据结构与软件结构。本概要设计说明书的目的就是进一步细化软件设计阶段得出的软件概貌，把它加工成在程序细节上非常接近与源程序开发的软件表示。

预期读者：软件测试员、程序开发员、软件分析员

1.2 背景

计算机视觉是人工智能的重要领域之一。在神经网络被应用在艺术领域之前，图像风格迁移的程序有一个共同的思路：分析某一种风格的图像，给那一种风格建立一个数学或者统计模型，再改变要做迁移的图像让它能更好的符合建立的模型。但一个很大的缺点：一个程序基本只能做某一种风格或者某一个场景。因此基于传统风格迁移研究的实际应用非常有限。

随着计算机运算能力的增加，以及算法效率的优化，将机器学习算法应用到艺术领域逐渐成为可能。运用神经网络方法进行图片风格迁移，可以方便用户，满足创作的需求，轻松地完成带有某种特定风格的新作品， 同时也可以进一步理解图像艺术风格的量化形式。

对于图片风格迁移的研究主要在 2012 年以后，也就是深度学习算法开始被广泛应用的时候。卷及神经网络为图片特征的提取提供了强有力的工具。我们使用成百上千的神经元用于提取特征，当图片中出现符合的特征时会激活相应的神经元，当神经网络识别出图片的特征后，可以通过对比两幅图片的相似图形特征，并将一幅图片的颜色特征应用到另一幅图片中，从而生成带有特定风格的新作品。

目前基于 Gatys 的论文，许多研究者实现了不同的图片风格迁移脚本，但真正将这项技术实现为应用的寥寥无几。据了解，相机软件中“艺术家”APP, “Adobe”公司的自拍软件等使用了滤镜来实现风格迁移，但仅限指定的风格。Python 开源库中 Prisma 提供了相关接口，但仅面向于开发者，没有编程能力的用户难以使用。

基于 Golnaz Ghiasi 等人的论文，风格迁移算法可以不必固定于某一种风格特征，而是可以针对任意风格进行自动迁移，并且运算时间被降至秒级甚至微妙级（受 CPU 和 GPU 性能的影响）。这一算法最早被作者用 Lua 语言实现为脚本，并发布在网络上，但非相关专业人士，尤其是设计领域人士，难以配置复杂的运行环境并使用此脚本去实现风格迁移。

本论文将此算法移植到应用平台，旨在为用户提供一个界面清洁，方便可用的 Web 应用，使用户能通过本应用进行快速的艺术创新，为自己的作品增添创意。

1.3 定义

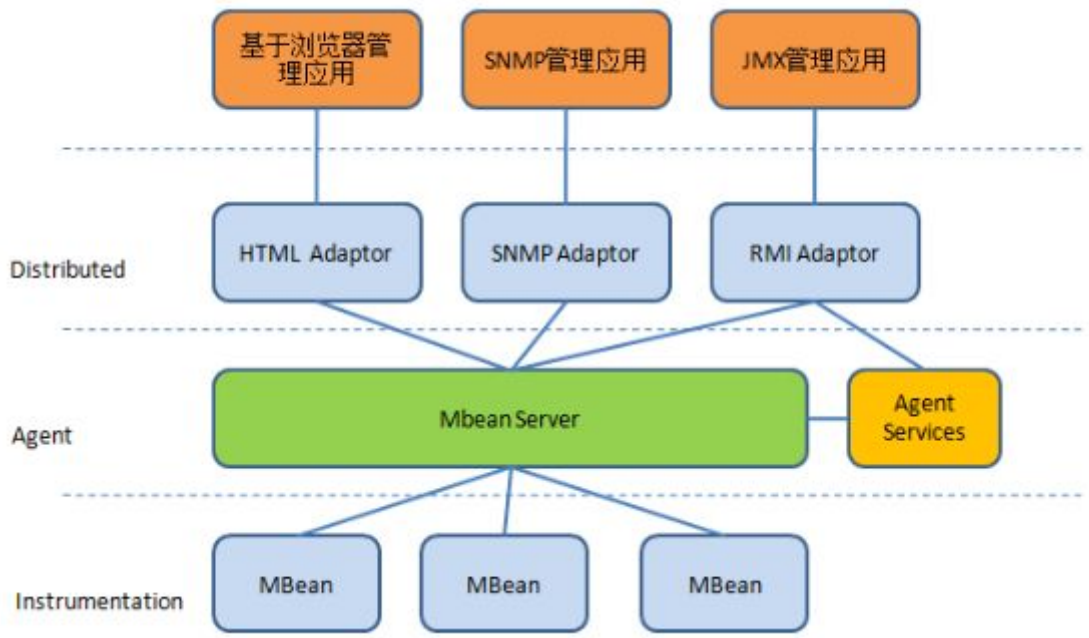
a. Mysql:系统服务器所使用的数据库关系系统（DBMS）。
b. SQL：一种用于访问查询数据库的语言
c. 事务流：数据进入模块后可能有多种路径进行处理。
d. 主键：数据库表中的关键域。值互不相同。
e. 外部主键：数据库表中与其他表主键关联的域。
f. ROLLBACK:数据库的错误恢复机制。
g. 缩写：

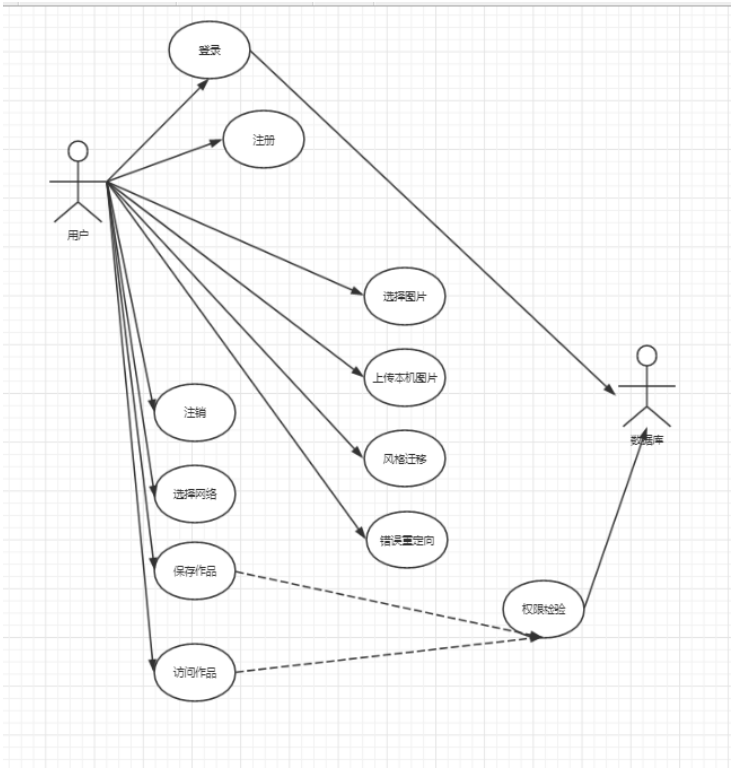
h. SQL：Structured Query Language(结构化查询语言)。
i. ATM：Asynchronous Transfer Mode (异步传输模式)。
j. UML：统一建模语言、是一套用来设计软件蓝图的标准建模语言，是一种从软件分析、设计到编写程序规范的标准化建模语言。

1.4 参考资料

a. 项目开发计划；
b. MyArtist 需求规格说明书；
c. MyArtist 需求清单
d. 《软件工程概论》 李存珠编著 南京大学计算机系出版 2001 年 8 月
e. 《软件工程——实践者的研究方法》Roger S Pressman 著

2. 系统的结构





3．模块设计说明

3.1 前端脚本实现

3.1.1 从本机上传文件及图片选择模块

```
/*
函数 setImages(): 从下拉菜单选择图片或者从本机读取图片

参数 element：
HTML 中图片元素的引用

参数 selectedValue：
图片对应下拉框选中的值，类型为字符串。
如果为图片，则为图片名称字符串；如果为本机上传图片，则为”file”

无返回值
*/
setImage(element, selectedValue) {
    if (selectedValue === 'file') {        //如果选中的是“选择本机文件”
        console.log('file selected');      //输出到控制台信息
```

```
        this.fileSelect.onchange = (evt) => {  //用户选中文件后
            const f = evt.target.files[0];
            const fileReader = new FileReader();
            fileReader.onload = ((e) => {
                element.src = e.target.result;
            });
            fileReader.readAsDataURL(f);        //调用 FileReader 读取文件作为图片元素数据源
            this.fileSelect.value = '';         //恢复选择状态到初始状态
        }
        this.fileSelect.click();
    }else {
        element.src = 'images/' + selectedValue + '.jpg'; //如果选中的是图片,则将图片作为数据源
    }
  }
```

```
/*
函数 setContent(): 从图片预览框选择内容图片

参数 picName：
图片对应的文件名称，类型为字符串。

无返回值
*/
        function setContent(picName) {
            var contentSelect = document.getElementById('content-select');
            contentSelect.value = picName;
            var contentImg = document.getElementById('content-img');
            contentImg.src = 'images/' + picName + '.jpg';
            document.getElementById('wait_pic').click();
        }
```

```
/*
函数 setStyle(): 从图片预览框选择风格图片

参数 picName：
图片对应的文件名称，类型为字符串。

无返回值
*/
function setStyle(picName) {
            var styleSelect = document.getElementById('style-select');
            styleSelect.value = picName;
```

```
        var styleImg = document.getElementById('style-img');
        styleImg.src = 'images/' + picName + '.jpg';
        document.getElementById('style_pic').click();
    }
```

3.1.2 图片预处理及载入模型

/*
函数 startStyling(): 选中风格图片、内容图片后点击“风格迁移”按钮会调用initializeStyleTransfer()初始化，然后再调用本函数处理。

无参数

无返回值
*/

```
async startStyling() {
    await tf.nextFrame();
    this.styleButton.textContent = '生成风格样式';          //风格迁移按钮名称改变
    await tf.nextFrame();
    //计算风格图片的风格
    let bottleneck = await tf.tidy(() => {
      return
this.styleNet.predict(tf.fromPixels(this.styleImg).toFloat().div(tf.scalar(255)).expandDims());
    })                                                    //使用载入的模型对图片处理获得风格
图片的特征向量
      //如果风格系数不为 1，那么内容图片也计算风格
    if (this.styleRatio !== 1.0) {
      this.styleButton.textContent = '生成风格样式';
      await tf.nextFrame();
      const identityBottleneck = await tf.tidy(() => {
        return
this.styleNet.predict(tf.fromPixels(this.contentImg).toFloat().div(tf.scalar(255)).expandDims());
      })                                                  //使用载入的模型对图片处理获得内容
图片的特征向量
        //tf.tidy()可以清空无用的内存空间
      const styleBottleneck = bottleneck;//风格图片的风格向量
      bottleneck = await tf.tidy(() => {
        const styleBottleneckScaled = styleBottleneck.mul(tf.scalar(this.styleRatio));
        const identityBottleneckScaled = identityBottleneck.mul(tf.scalar(1.0-this.styleRatio));
        return styleBottleneckScaled.addStrict(identityBottleneckScaled)
      })
        //风格图片的风格*风格系数+内容图片的风格*（1-风格系数） 即求两幅图片的风
```

```
格的加权平均
        styleBottleneck.dispose();
        identityBottleneck.dispose();
        //释放内存
      }
      this.styleButton.textContent = '正在迁移风格';
      await tf.nextFrame();
      const stylized = await tf.tidy(() => {
        return
this.transformNet.predict([tf.fromPixels(this.contentImg).toFloat().div(tf.scalar(255)).expandDims
(), bottleneck]).squeeze();
      })
        //调用载入的风格迁移网络,计算风格化后的图片张量
      await tf.toPixels(stylized, this.stylized);
      //转换为像素
      bottleneck.dispose();  //释放内存
      stylized.dispose();
    }
```

```
/*
函数 startStyling(): 选中风格图片、内容图片后点击“风格迁移”按钮会调用此函数处理。

无参数

无返回值
*/
```

```
initializeStyleTransfer() {
      // 风格迁移初始化
      this.contentImg = document.getElementById('content-img');
      this.contentImg.onerror = () => {              //如果内容图片载入失败，弹出错误信息
        alert("Error loading " + this.contentImg.src + ".");
      }
      this.styleImg = document.getElementById('style-img');
      this.styleImg.onerror = () => {                //如果风格图片载入失败，弹出错误信息
        alert("Error loading " + this.styleImg.src + ".");
      }
      this.stylized = document.getElementById('stylized');
      this.styleRatio = 1.0

      // Initialize buttons
      this.styleButton = document.getElementById('style-button');
      this.styleButton.onclick = () => {
        this.disableStylizeButtons();
```

```
        this.startStyling().finally(() => {       //调用风格迁移函数 startStyling()并等待结束后再次激活按钮
            this.enableStylizeButtons();
        });
    };
}
```

```
/*
模型选择模块主要代码
*/
```

```
    this.modelSelectStyle = document.getElementById('model-select-style');
    this.modelSelectStyle.onchange = (evt) => {
        if (evt.target.value === 'mobilenet') {              //用户选中快速特征提取网络'mobilenet'
            this.disableStylizeButtons();
            this.loadMobileNetStyleModel().then(model => {
                this.styleNet = model;
            }).finally(() => this.enableStylizeButtons());  //载入完成后再次激活按钮
        } else if (evt.target.value === 'inception') {       //用户选中高质量特征提取网络'inception'
            this.disableStylizeButtons();
            this.loadInceptionStyleModel().then(model => {
                this.styleNet = model;
            }).finally(() => this.enableStylizeButtons());  //载入完成后再次激活按钮
        }
    }

    this.modelSelectTransformer = document.getElementById('model-select-transformer');
    this.modelSelectTransformer.onchange = (evt) => {
        if (evt.target.value === 'original') {               //用户选中原始风格迁移网络 Origin
            this.disableStylizeButtons();
            this.loadOriginalTransformerModel().then(model => {
                this.transformNet = model;
            }).finally(() => this.enableStylizeButtons());  //载入完成后再次激活按钮
        } else if (evt.target.value === 'separable') {           //用户选中快速风格迁移网络 Separable_conv2d
            this.disableStylizeButtons();
            this.loadSeparableTransformerModel().then(model => {
                this.transformNet = model;
            }).finally(() => this.enableStylizeButtons());  //载入完成后再次激活按钮
        }
    }
```

3.1.3 作品图片上传模块

```
/*
函数 uploadimage() 上传用户图片作品到服务器上
返回值：0
*/
    function uploadimage() {
                var canvas = document.getElementById("stylized");
                var data = canvas.toDataURL("image/png", 1);     //将 canvas 转化为数据格式，保留原分辨率
                if (data.length === 1594) {     //如果作品图片为空白（canvas 大小为 1594 字节），弹出提示
                    alert("请先迁移风格");
                    return 0;
                }

                data = data.split(',')[1];                       //取出数据中的图片内容部分
                data = window.atob(data);                        //转换为网络流格式
                var ia = new Uint8Array(data.length);
                for (var i = 0; i < data.length; i++) {
                    ia[i] = data.charCodeAt(i);                  //迭代读取数据内容
                }
                var blob = new Blob([ia], {type: "image/png"});  //封装进 Blob 数据结构中，方便服务器解析

                var fd = new FormData();
                fd.append("uploadimage", blob);          //将 Blob 封装进 FormData 数据结构中，方便发送数据
                $.ajax({
                    url: "/uploadimage.action",
                    type: 'post',                        //发送 post 请求
                    processData: false,
                    contentType: false,
                    data: fd,
                    dataType: 'json',
                    success: function (data) {
                        if (data === 0) {                //返回值为 0 则服务器保存失败，弹出提示。
                            alert("server error!");
                        } else {
                            alert("上传成功！");     //其他返回值则保存成功，弹出提示。
```

```
                    }
                },
                error: function (jqXHR, textStatus, errorThrown) {
                    alert(textStatus + "---" + errorThrown);  //若浏览器调用出现问题，
则弹出提示。
                }
            }
        );
        return 0;
    }
```

3.1.4 作品图片展示模块

```
/*
当“我的图库”页面载入时，自动向服务器发送读取图片列表请求。
*/
window.onload = function () {
    $.ajaxSettings.async = false;
    $.get                    //发出 GET 请求
    ("/getMyImage.action",
        {},
        function (result) {
            if (result === null) {          //如果返回空值，则弹出加载失败提示
                alert("数据加载失败！");
            }
            else if (result.length === 0) {  //如果图片列表长度为 0,则提示没有图
片
                alert("您目前没有图片");
            }
            else {
                for (var i = 0; i < result.length; i++) {
                    if (result[i].filepath !== null && result[i].filepath !==
undefined) {
                        addImageToSlide(result[i].filepath, result[i].id);    // 将 图
片加载到幻灯片
                        addImageToBelow(result[i].filepath, result[i].id)   // 将 图
片加载到底部预览框
                    }
                }
                jQuery(function () {
                    jQuery('#camera_wrap').camera({
                        height: '400px',
                        loader: 'bar',
```

```
                pagination: false,
                thumbnails: true
            });
        });
        var thumb = document.getElementsByClassName("camera_thumb");
        console.log(thumb.length);
        for (i = 0; i < thumb.length; i++) {
            //调节幻灯片大小，以保持一致
            thumb[i].height = "75";
            thumb[i].width = "100";
        }
    }
}
);
```

3.1.5 注册登录注销模块

```
/*
登录模块函数 login()

无参数

无返回值
*/
function login()
    {
        var pswd = $("#pswd").val();
        var userId = $("#userId").val();      //获取用户名和密码
        $.post("/login.action",
            {
                id: userId,
                password: pswd              //发出 POST 请求
            },
            function (result) {
                if(result===0)        //如果返回值为 0 则提示用户名或密码错误
                {
                    alert("用户名或密码错误，请重试！");
                }
                else{
                    window.location.replace("index.html"); //若正确无误则调转到风格迁移主页面
                }
```

```
            });
        }
```

```
/*
注册模块函数 register()

无参数

无返回值
*/
function register() {
        var pswd = $("#password").val();
        var pswd2 = $("#confirm_password").val();
        var userName = $("#userName").val();
        var userId = $("#userId").val();
        var email = $("#email").val();
        var sex = $('input[name=a]:checked').val();
        var check=$("#agree").is(":checked");
        if(!check){
            alert("请同意协议后再点击注册！");          //未同意则提示错误
            return;
        }
        if(userName.length<2){
            alert("姓名长度太短!");                    //名字过短则提示错误
            return;
        }
        if(pswd!=pswd2){
            alert("两次密码不一致!");                  //两次密码不一致则提示错误
            return;
        }
        if(email.length<5){
            alert("请输入正确的邮箱！");               //有限长度过短则提示错误
            return;
        }
        if(userId.length<2){
            alert("请按要求输入 ID！");                //ID 长度过短则提示错误
            return;
        }
        $.post("/register.action",                     //POST 提交注册请求
            {
                id: userId,
                username:userName,
                password: pswd,
                email:email,
```

```
                    sex:sex
                },
                function (status) {
                    if (status === 0) {
                        alert("ID 已被注册，请重新输入!");          //返回值为 0 则代表
ID 已被注册,弹出提示
                    }
                    else {
                        alert("注册成功!请登录!");                  //其他返回值则提示
注册成功并跳转
                        window.location.replace("login.html");
                    }
                }
            );
        }
```

```
    /*
    用户注销函数 logout()

    无参数

    无返回值
    */
    function logout() {
                $.post("/logout.action",              //POST 提交注销请求
                    {},
                    function (result) {
                        if (result === 0) {          //返回值为 0 则提示注销错误
                            alert("退出过程出现错误");
                        }
                        else {
                            alert("退出成功");        //其他返回值则提示成功并跳转到
登录页面
                            window.location.replace("/login_user.html");
                        }
                    });
            }
```

3.2 服务器端服务实现

3.2.1 注册服务模块

```
/*
```

UserController.java
处理注册请求的控制器函数 register()
接收 POST 请求
参数 user 为 User 类对象，参数 session 为 HTTP 协议回话对象
返回 1 则成功，返回 0 则失败
*/

```
@RequestMapping (value = "/register.action",method = RequestMethod.POST)
    @ResponseBody
    public int register(User user,HttpSession session){
        if(user==null){
            return 0;  //如果接受到的用户对象为空指针,则直接返回 0,停止注册.
        }
        logger.info("controller-user-register:"+user.toString());
        try{
            int status=userService.register(user)==1?1:0; //调用注册 service，返回值传送到给
页面 view 层
            return   status;
        }catch (Exception e){
            System.out.println(e.getMessage());      //如果捕捉到异常,则打印异常信息.
            return 0;
        }
    }
```

/*
UserService.java
服务器内部对注册信息处理的函数 register()
通过 controller 调用
参数 record 为 User 类对象
返回 1 则成功，返回 0 则失败
*/

```
    public int register(User record) {
            System.out.println("serviceimpl:"+record.getId());
            if(userMapper.selectByPrimaryKey(record.getId())!=null)    //如果存在相同 ID 已
注册用户
            {
                logger.info("already exits, fail to register:"+record.getId());
                return 0;                                              //返回注册错误码 0
            }
            else
            {
                try{
                    int result=userMapper.insertSelective(record);
                    if (result==1){
```

```
                    return 1;                                   //注册成功返回 1
                }
                else{
                    return 0;                                   //注册而失败返回 0
                }
            }catch (Exception e){
                logger.error(e.getMessage());
                return 0;                                       //发生异常则打印信息并
返回 0
            }
        }
    }
```

3.2.1 登录服务模块

```
/*
UserController.java
处理登录请求的控制器函数 login()
接收 POST 请求
参数 user 为 User 类对象，参数 session 为 HTTP 协议回话对象
返回 1 则成功，返回 0 则失败
*/
    @RequestMapping (value = "/login.action")
        @ResponseBody
        public int login(/*@RequestBody*/ User user,    HttpSession session)
        {
            User sessionUser=(User)session.getAttribute("user");
            String userId=user.getId();
            System.out.println("try login:"+user);
            System.out.println("tri loginsession:"+sessionUser);
            if(user.getId()==null||user.getPassword()==null)            //如果请求数据不完
整=，报错返回 0
            {
                System.out.println("some is null");
                return 0;
            }
            if(sessionUser==null)                                       //如果用户未登录
            {
                System.out.println("session==null");
                int result=userService.login(user);                     //进行登录操作
                if(result==1)
                {
                    User findUser=userService.getUserById(userId);
```

```
                session.setAttribute("user", findUser);
                return 1;                                    //登录成功则添加 session 信息
返回 1
            }
            else{
                return 0;                                    //登录失败则返回 0
            }
        }else{
            System.out.println("user reloginUser:"+sessionUser);   //如果用户已登录,直
接返回 1
            return 1;
        }
    }
```

```
/*
UserService.java
服务器内部对登录信息处理的函数 login()
通过 controller 调用
参数 record 为 User 类对象
返回 1 则成功，返回 0 则失败
*/
```

```
public int login(User user){
    User user1=userMapper.selectByPrimaryKey(user.getId());  //读取用户信息
    try{
        String pswd=user1.getPassword();
        if(pswd.equals(user.getPassword())){                 //对比密码信息是否相同
            return 1;                                        //相同则返回 1 登录成功
        }
        else{
            return 0;                                        //不相同则返回 0 登录失败
        }
    }catch (Exception e)
    {
        logger.error(e.getMessage());                        //出现异常则打印信息并
返回 0 登录失败
        return 0;
    }
}
```

3.2.1 注销服务模块

```
/*
UserController.java
处理注销请求的控制器函数 logout()
```

接收 POST 请求
参数 user 为 User 类对象，参数 session 为 HTTP 协议回话对象
返回值为 1
*/

```
    @RequestMapping (value = "/logout.action",method = RequestMethod.POST)
    @ResponseBody
    public    String logout(HttpSession session){
        System.out.println("controller-user-logout:");
        session.invalidate();                    //直接将 session 会话失效即可
        return "1";
    }
```

3.2.1 图片上传服务模块

```
/*
ImageController.java
处理图片上传请求的控制器函数 uploadImage()
接收 POST 请求
参数 uploadimage 为 MultipartFile 类对象，参数 session 为 HTTP 协议回话对象
返回 1 则成功，返回-1 则失败,0 代表服务器异常
*/
@RequestMapping(value = "/uploadimage.action",method = RequestMethod.POST)
    public int uploadImage(@RequestParam("uploadimage") MultipartFile
uploadimage,HttpSession session){
        try {
            String path =
ClassUtils.getDefaultClassLoader().getResource("").getPath()+"static/upload";  //图片保存路径
以资源文件目录为基准
            User sessionUser=(User)session.getAttribute("user");        //获取用户当前登录
状态
            if (sessionUser==null)
            {
                return -1;                                          //如果用户未登录,返回-1  上传
失败
            }
            MultipartFile uploadFile = uploadimage;

            Image image = new Image();
            SimpleDateFormat df = new SimpleDateFormat("yyyy 年 MM 月 dd 日 HH 时 mm
分 ss 秒");//设置日期格式
            String dateString=df.format(new Date());// new Date()为获取当前系统时间
            long t = System.currentTimeMillis();
            Random rd = new Random(t);              //为文件名后增加随机数字
```

```
            String filename = "用户"+sessionUser.getId()+"-"+dateString+"-"+rd.nextInt()+".png";
            String pathAndName=path +"/" +filename;   //保存文件目录及名称

            InputStream is = uploadFile.getInputStream();
            // 如果服务器已经存在和上传文件同名的文件，则输出提示信息
            File tempFile = new File(pathAndName);
            File fileParent = tempFile.getParentFile();          //获得文件目录地址
            if(!fileParent.exists()){                            //如果目录不存在,则创建文件夹
                fileParent.mkdirs();
            }
            if (tempFile.exists()) {
                boolean delResult = tempFile.delete();           //如果该文件已存在,则覆盖此文件
                System.out.println("删除已存在的文件：" + delResult);
            }
            // 开始保存文件到服务器
            if (!filename.equals("")) {
                FileOutputStream fos = new FileOutputStream(pathAndName);   //接收文件
                byte[] buffer = new byte[8192]; // 每次读 8K 字节
                int count = 0;
                // 开始读取上传文件的字节，并将其输出到服务端的上传文件输出流中
                while ((count = is.read(buffer)) > 0) {
                    fos.write(buffer, 0, count); // 向服务端文件写入字节流
                }
                fos.close(); // 关闭 FileOutputStream 对象
                is.close(); // InputStream 对象


                image.setId(sessionUser.getId()+"-"+dateString+"-"+rd.nextInt());
                image.setFilepath("upload/"+filename);
                image.setUserId(sessionUser.getId());          //为文件对象设置基本属性

                imageService.insertImage(image);               //调用服务,将文件信息插入数据库中
                System.out.println("file name:"+path);

            }
            return 1;                          //操作正常,则返回 1 表示上传成功
        } catch (IOException f) {
            f.printStackTrace();
            return 0;                          //出现异常则打印异常信息并返回错误码 0
        }
```

```
    }
```

```
/*
ImageService.java
服务器内部对图片插入处理的函数 insertImage()
通过 controller 调用
参数 image 为 Image 类对象
返回 1 则成功，返回 0 则失败
*/
public int insertImage(Image image)
    {
        try {
            imageMapper.insertSelective(image);    //向数据库插入数据
            return 1;
        }catch (Exception e){
            e.printStackTrace();                    //插入异常则打印信息返回 0
            return 0;
        }
    }
```

3.2.1 图片库读取模块

```
/*
ImageController.java
处理图片上传请求的控制器函数 getMyImage()
接收 GET 请求
参数 session 为 HTTP 协议回话对象
返回图片类的列表 myImageList，类型为 List<Image>
*/
    @RequestMapping(value = "/getMyImage.action",method = RequestMethod.GET)
    public List<Image> getMyImage(HttpSession session){
        User user=(User)session.getAttribute("user");                    //从 session 中读
取用户信息
        String userid=user.getId();
        List<Image> myImageList=imageService.getMyImage(userid);  //调用服务获得用户
图库
        return myImageList;                                              //返回图片列表
    }
/*

ImageService.java
服务器内部对图片信息读取的函数 getMyImage()
通过 controller 调用
```

参数 userid 为 String 类对象
返回 1 则成功，返回 0 则失败
*/

```
        public List<Image> getMyImage(String userid) {
            List<Image> imageList=imageMapper.selectByUserid(userid);    //通过数据库查询图片数据
            System.out.println("获取图片中："+imageList.size());
            return imageList;                                              //返回图片列表
        }
```

3.2.1 权限控制模块

/*
WebConfig.java
对拦截器状况进行配置，注册 AuthInterceptor 到服务器拦截器中
*/

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        //设置对“/”的请求映射到 index
        //如果没有数据返回到页面，没有必要用控制器方法对请求进行映射
        registry.addViewController("/").setViewName("index.html");
    }
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        //SpringMVC 下，拦截器的注册需要排除对静态资源的拦截(*.css,*.js)
        //SpringBoot 已经做好了静态资源的映射，因此我们无需任何操作
        registry.addInterceptor(new AuthInterceptor()).addPathPatterns("/**.html","/uploadimage.action")
                .excludePathPatterns("/login_user.html","/register_user.html");    //除了注册登录页面外,其他页面以及上传图片服务时要经过权限检查
    }
}
```

/*
AuthInterceptor.java
拦截器的具体实现
启动了页面处理前置拦截(preHandle)
重定向页面以及权限检查
*/

```
@Override
    public boolean preHandle(HttpServletRequest request,
```

```
                                    HttpServletResponse response, Object handler) throws
Exception {
        System.out.println("==============  执  行  顺  序  :  1  、
preHandle================");
        String requestUri = request.getRequestURI();
        String contextPath = request.getContextPath();
        String url = requestUri.substring(contextPath.length());
        List<Integer> errorCodeList = Arrays.asList(404, 403, 500, 501);//访问非法页面的错
误列表
        log.info(""+response.getStatus()+"\n");
        if (errorCodeList.contains(response.getStatus())) {
            log.info("error page");
            response.sendRedirect("/index.html");                    //如果访问页面状态在列
表中,重定向到主页
            return false;
        }
        log.info("requestUri:"+requestUri);
        log.info("contextPath:"+contextPath);
        System.out.println("url:"+url);
        User user =  (User)request.getSession().getAttribute("user");          // 获 得 用 户
session 的登录状态
        if(user == null){
            log.info("Interceptor：跳转到 login 页面！");
            //request.getRequestDispatcher("/static/login.html").forward(request, response);
            response.sendRedirect(request.getContextPath()+"/login_user.html");  //如果用户
没有登录则跳转主页
            return false;
        }else
            return true;                                                  //否则放行
    }
```