3.13 个人信息查询接口

3.13.1 需求

进入个人中心的时候需要能够查看当前用户信息

3.13.2 接口设计

请求方式	请求地址	请求头
GET	/user/userInfo	需要token请求头

不需要参数

响应格式:

```
{
    "code":200,
    "data":{

"avatar":"https://gimg2.baidu.com/image_search/src=http%3A%2F%2Fi0.hdslb.com%2Fbfs%2Farticle%2F3bf9c263bc0f2ac5c3a7feb9e218d07475573ec8.gi",
        "email":"23412332@qq.com",
        "id":"1",
        "nickName":"sg333",
        "sex":"1"
    },
    "msg":"操作成功"
}
```

3.13.3 代码实现

UserController

```
@RestController
@RequestMapping("/user")
public class UserController {

    @Autowired
    private IUserService userService;

    @GetMapping("/userInfo")
    public ResponseResult userInfo() {
        return userService.userInfo();
    }
}
```

UserService增加方法定义

```
public interface IUserService extends IService<User> {

    ResponseResult userInfo();
}
```

UserServiceImpl实现userInfo方法

```
    @Override
    public ResponseResult userInfo() {
        //获取当前用户id
        Long userId = SecurityUtils.getUserId();
        //根据用户id查询用户信息
        User user = getById(userId);
        //封装成UserInfoVo
        UserInfoVo vo = BeanCopyUtils.copyBean(user,UserInfoVo.class);
        return ResponseResult.okResult(vo);
    }
```

SecurityConfig配置该接口必须认证后才能访问

```
   @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                //关闭csrf
                .csrf().disable()
                //不通过Session获取SecurityContext

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and()
                .authorizeRequests()
                // 对于登录接口 允许匿名访问
                .antMatchers("/login").anonymous()
                //注销接口需要认证才能访问
                .antMatchers("/logout").authenticated()
                //发表评论需要认证才能访问
                .antMatchers("/comment").authenticated()
                //个人信息接口必须登录后才能访问
                .antMatchers("/user/userInfo").authenticated()
                // 除上面外的所有请求全部不需要认证即可访问
                .anyRequest().permitAll();

        //配置异常处理器
        http.exceptionHandling()
                .authenticationEntryPoint(authenticationEntryPoint)
                .accessDeniedHandler(accessDeniedHandler);
        //关闭默认的注销功能
        http.logout().disable();
        //把jwtAuthenticationTokenFilter添加到SpringSecurity的过滤器链中
```

```
        http.addFilterBefore(jwtAuthenticationTokenFilter,
UsernamePasswordAuthenticationFilter.class);
        //允许跨域
        http.cors();
    }
```

3.14 头像上传接口

3.14.1 需求

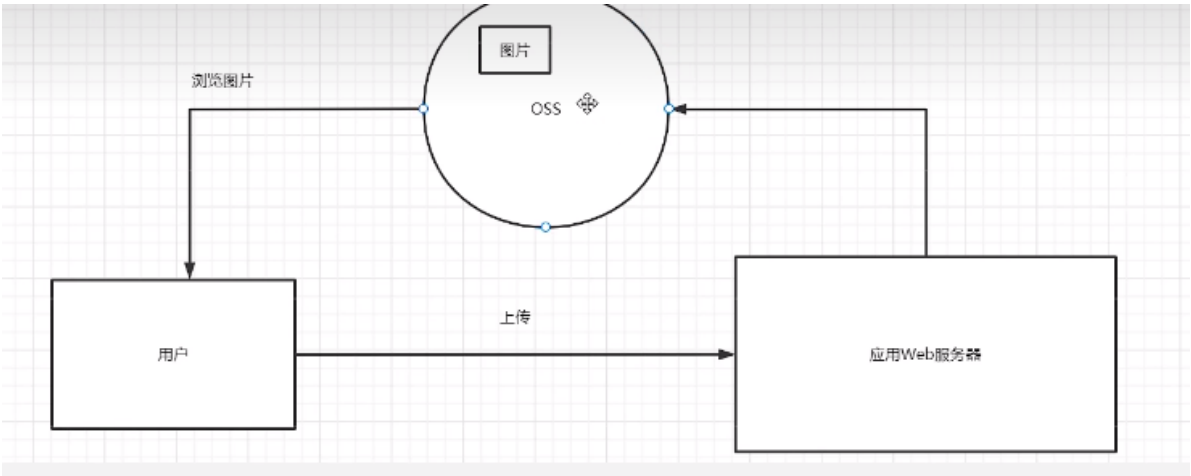
在个人中心点击编辑的时候可以上传头像图片。上传完头像后，可以用于更新个人信息接口。

3.14.2 OSS

3.14.2.1 为什么要使用OSS

因为如果把图片视频等文件上传到自己的应用的Web服务器，在读取图片的时候会占用比较多的资源。影响应用服务器的性能。

所以我们一般使用OSS(Object Storage Service对象存储服务)存储图片或视频。



3.14.2.2 七牛云基本使用测试

注册认证

创建存储空间


资源管理
存储空间 1
+ 新建存储空间
加速域名 1
+ 添加加速域名
全站加速域名 0
+ 添加全站加速域名
SSL 证书 0
+ 新增 SSL 证书
多媒体工作流 0
+ 新建工作流
多媒体任务触发器 0
+ 新建任务触发器
容器轻应用平台 0
+ 新建应用
存储空间增量审核任务 0
+ 新建增量审核任务
直播空间 0
+ 新建直播空间
QVS 空间 0
+ 新建 QVS 空间
云主机实例 0
+ 新建云主机实例
云数据库 0
+ 新建云数据库
高防 DDOS IP 0
+ 新建高防 DDOS IP


新建存储空间
存储空间名称：
ptu-blog2
存储空间名称不允许重复，遇到冲突请更换名称。
名称格式为 3 ~ 63 个字符，可以包含小写字母、数字、短划线，且必须以小写字母或者数字开头和结尾。
存储区域：
华东-浙江
华北-河北
华南-广东
北美-洛杉矶
亚太-新加坡（原东南亚）
华东-浙江2
HOT
此空间将会在 华南-广东 创建。
访问控制：
公开
私有
公开和私有仅对 Bucket 的读文件生效，修改、删除、写入等对 Bucket 的操作均需要拥有者的授权才能进行操作。

秘钥

3.14.2 接口设计

请求方式	请求地址	请求头
POST	/upload	需要token

参数:

img,值为要上传的文件

请求头:

Content-Type ： multipart/form-data;

响应格式:

```
{
    "code": 200,
    "data": "文件访问链接",
    "msg": "操作成功"
}
```

3.14.3 代码实现

```
@RestController
public class UploadController {
    @Autowired
    private UploadService uploadService;

    @PostMapping("/upload")
    public ResponseResult uploadImg(MultipartFile img){
        return uploadService.uploadImg(img);
    }
}
```

```
public interface UploadService {
    ResponseResult uploadImg(MultipartFile img);
}
```

七牛云上传代码编写

①添加依赖

```
        <dependency>
            <groupId>com.qiniu</groupId>
            <artifactId>qiniu-java-sdk</artifactId>
            <version>[7.7.0, 7.7.99]</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-configuration-processor</artifactId>
            <optional>true</optional>
        </dependency>
```

②添加配置信息

application.yml

```
oss:
  accessKey: xxxx
  secretKey: xxxx
  bucket: ptu-blog
```

3. 复制修改案例代码

```
@Service
@Data
@ConfigurationProperties(prefix = "oss")
public class OSSUploadServiceImpl implements UploadService {
    private String accessKey;
    private String secretKey;
    private String bucket;
    @Override
```

```
    public ResponseResult uploadImg(MultipartFile img) {
        //判断文件类型
        //获取原始文件名
        String originalFilename = img.getOriginalFilename();
        //对原始文件名进行判断
        if(!originalFilename.endsWith(".png")){
            throw new SystemException(AppHttpCodeEnum.FILE_TYPE_ERROR);
        }

        //如果判断通过上传文件到OSS
        String filePath = PathUtils.generateFilePath(originalFilename);
        String url = uploadOss(img,filePath);//  2099/2/3/wqeqeqe.png
        return ResponseResult.okResult(url);
    }

    private String uploadOss(MultipartFile img, String filePath) {
        //构造一个带指定 Region 对象的配置类
        Configuration cfg = new Configuration(Region.autoRegion());
        cfg.resumableUploadAPIVersion =
Configuration.ResumableUploadAPIVersion.V2;// 指定分片上传版本
//...其他参数参考类注释

        UploadManager uploadManager = new UploadManager(cfg);
//...生成上传凭证，然后准备上传
//        String accessKey = "your access key";
//        String secretKey = "your secret key";
//        String bucket = "your bucket name";

//默认不指定key的情况下，以文件内容的hash值作为文件名
        String key = filePath;

        try {
//            byte[] uploadBytes = "hello qiniu cloud".getBytes("utf-8");
//            ByteArrayInputStream byteInputStream=new
ByteArrayInputStream(uploadBytes);
            InputStream byteInputStream = img.getInputStream();
            Auth auth = Auth.create(accessKey, secretKey);
            String upToken = auth.uploadToken(bucket);

            try {
                Response response =
uploadManager.put(byteInputStream,key,upToken,null, null);
                //解析上传成功的结果
                DefaultPutRet putRet = new
Gson().fromJson(response.bodyString(), DefaultPutRet.class);
                System.out.println(putRet.key);
                System.out.println(putRet.hash);
                return "http://rv6a8n5c7.hn-bkt.clouddn.com/"+key;
            } catch (QiniuException ex) {
                Response r = ex.response;
                System.err.println(r.toString());
                try {
                    System.err.println(r.bodyString());
                } catch (QiniuException ex2) {
                    //ignore
                }
            }
        } catch (UnsupportedEncodingException ex) {
```

```
            //ignore
        } catch (IOException e) {
            e.printStackTrace();
        }
        return "abc";
    }
}
```

PathUtils

```
public class PathUtils {

    public static String generateFilePath(String fileName){
        //根据日期生成路径   2023/5/25/
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd/");
        String datePath = sdf.format(new Date());
        //uuid作为文件名
        String uuid = UUID.randomUUID().toString().replaceAll("-", "");
        //后缀和文件后缀一致
        int index = fileName.lastIndexOf(".");
        // test.jpg -> .jpg
        String fileType = fileName.substring(index);
        return new
StringBuilder().append(datePath).append(uuid).append(fileType).toString();
    }
}
```

3.15 更新个人信息接口

3.15.1 需求

在编辑完个人资料后点击保存会对个人资料进行更新。

3.15.2 接口设计

请求方式	请求地址	请求头
PUT	/user/userInfo	需要token请求头

参数

请求体中json格式数据：

```
{
    "avatar":"https://****/2023/05/25/948597e164614902ab1662ba8452e106.png",
    "email":"test@qq.com",
    "id":"3",
    "nickName":"ptu",
    "sex":"1"
}
```

响应格式:

```
{
    "code":200,
    "msg":"操作成功"
}
```

3.15.3 代码实现

UserController

```
    @PutMapping("/userInfo")
    public ResponseResult updateUserInfo(@RequestBody User user){
        return userService.updateUserInfo(user);
    }
```

UserService

```
ResponseResult updateUserInfo(User user);
```

UserServiceImpl

```
    @Override
    public ResponseResult updateUserInfo(User user) {
        updateById(user);
        return ResponseResult.okResult();
    }
```

3.16 用户注册

3.16.1 需求

要求用户能够在注册界面完成用户的注册。要求用户名，昵称，邮箱不能和数据库中原有的数据重复。如果某项重复了注册失败并且要有对应的提示。并且要求用户名，密码，昵称，邮箱都不能为空。

注意:密码必须密文存储到数据库中。

3.16.2 接口设计

请求方式	请求地址	请求头
POST	/user/register	不需要token请求头

参数

请求体中json格式数据:

```
{
  "email": "string",
  "nickName": "string",
  "password": "string",
  "userName": "string"
}
```

响应格式:

```
{
    "code":200,
    "msg":"操作成功"
}
```

3.16.3 代码实现

UserController

```
    @PostMapping("/register")
    public ResponseResult register(@RequestBody User user){
        return userService.register(user);
    }
```

UserService

```
ResponseResult register(User user);
```

UserServiceImpl

```
@Autowired
    private PasswordEncoder passwordEncoder;
    @Override
    public ResponseResult userInfo() {
        //获取当前用户id
        Long userId = SecurityUtils.getUserId();
        //根据用户id查询用户信息
        User user = getById(userId);
        //封装成UserInfoVo
        UserInfoVo vo = BeanCopyUtils.copyBean(user,UserInfoVo.class);
```

```
        return ResponseResult.okResult(vo);
    }

    @Override
    public ResponseResult updateUserInfo(User user) {
        updateById(user);
        return ResponseResult.okResult();
    }

    @Override
    public ResponseResult register(User user) {
        //对数据进行非空判断
        if(!StringUtils.hasText(user.getUserName())){
            throw new SystemException(AppHttpCodeEnum.USERNAME_NOT_NULL);
        }
        if(!StringUtils.hasText(user.getPassword())){
            throw new SystemException(AppHttpCodeEnum.PASSWORD_NOT_NULL);
        }
        if(!StringUtils.hasText(user.getEmail())){
            throw new SystemException(AppHttpCodeEnum.EMAIL_NOT_NULL);
        }
        if(!StringUtils.hasText(user.getNickName())){
            throw new SystemException(AppHttpCodeEnum.NICKNAME_NOT_NULL);
        }
        //对数据进行是否存在的判断
        if(userNameExist(user.getUserName())){
            throw new SystemException(AppHttpCodeEnum.USERNAME_EXIST);
        }
        if(emailExist(user.getEmail())){
            throw new SystemException(AppHttpCodeEnum.EMAIL_EXIST);
        }
        //...
        //对密码进行加密

        String encodePassword = passwordEncoder.encode(user.getPassword());
        user.setPassword(encodePassword);
        //存入数据库
        save(user);
        return ResponseResult.okResult();
    }

    private boolean emailExist(String email) {
        LambdaQueryWrapper<User> queryWrapper = new LambdaQueryWrapper<>();
        queryWrapper.eq(User::getEmail,email);
        return count(queryWrapper) >0;
    }

    private boolean userNameExist(String userName) {
        LambdaQueryWrapper<User> queryWrapper = new LambdaQueryWrapper<>();
        queryWrapper.eq(User::getUserName,userName);
        return count(queryWrapper) >0;
    }
```

```
public enum AppHttpCodeEnum {
    // 成功
    SUCCESS(200,"操作成功"),
    // 登录
```

```
    NEED_LOGIN(401,"需要登录后操作"),
    NO_OPERATOR_AUTH(403,"无权限操作"),
    SYSTEM_ERROR(500,"出现错误"),
    USERNAME_EXIST(501,"用户名已存在"),
     PHONENUMBER_EXIST(502,"手机号已存在"), EMAIL_EXIST(503, "邮箱已存在"),
    REQUIRE_USERNAME(504, "必需填写用户名"),
    CONTENT_NOT_NULL(506, "评论内容不能为空"),
    FILE_TYPE_ERROR(507, "文件类型错误，请上传png文件"),
    USERNAME_NOT_NULL(508, "用户名不能为空"),
    NICKNAME_NOT_NULL(509, "昵称不能为空"),
    PASSWORD_NOT_NULL(510, "密码不能为空"),
    EMAIL_NOT_NULL(511, "邮箱不能为空"),
    NICKNAME_EXIST(512, "昵称已存在"),
    LOGIN_ERROR(505,"用户名或密码错误");
    int code;
    String msg;

    AppHttpCodeEnum(int code, String errorMessage){
        this.code = code;
        this.msg = errorMessage;
    }

    public int getCode() {
        return code;
    }

    public String getMsg() {
        return msg;
    }
}
```

3.17 AOP实现日志记录

3.17.1 需求

需要通过日志记录接口调用信息。便于后期调试排查。并且可能有很多接口都需要进行日志的记录。

接口被调用时日志打印格式如下：

```
  : ==============Start==============
  : URL             : http://localhost:7777/user/userInfo
  : BusinessName    : 更新用户信息接口
  : HTTP Method     : PUT
  : Class Method    : com.my.blog.controller.UserController.updateUserInfo
  : IP              : 0:0:0:0:0:0:0:1
  : Request Args    : User{id = 4, userName = null, nickName = 软工, password
```

```
    : Response Args  : {"code":200,"msg":"操作成功"}
    : =======End=======
```

相当于是对原有的功能进行增强。并且是批量的增强，这个时候就非常适合用AOP来进行实现。

3.17.3 代码实现 自定义注解

日志打印格式

```
        log.info("=======Start=======");
        // 打印请求 URL
        log.info("URL            : {}",);
        // 打印描述信息
        log.info("BusinessName   : {}", );
        // 打印 Http method
        log.info("HTTP Method    : {}", );
        // 打印调用 controller 的全路径以及执行方法
        log.info("Class Method   : {}.{}", );
        // 打印请求的 IP
        log.info("IP             : {}",);
        // 打印请求入参
        log.info("Request Args   : {}",);
        // 打印出参
        log.info("Response       : {}", );
        // 结束后换行
        log.info("=======End=======" + System.lineSeparator());
```

```
@Before("customerJoinPointerExpression()")
public void beforeMethod(JoinPoint joinPoint){
    joinPoint.getSignature().getName(); // 获取目标方法名
    joinPoint.getSignature().getDeclaringType().getSimpleName(); // 获取目标方法所属类的简单类名
    joinPoint.getSignature().getDeclaringTypeName(); // 获取目标方法所属类的类名
    joinPoint.getSignature().getModifiers(); // 获取目标方法声明类型(public、private、protected)
    Object[] args = joinPoint.getArgs(); // 获取传入目标方法的参数，返回一个数组
    joinPoint.getTarget(); // 获取被代理的对象
    joinPoint.getThis(); // 获取代理对象自己
}

// 获取目标方法上的注解
private <T extends Annotation> T getMethodAnnotation(ProceedingJoinPoint joinPoint, Class<T> c
    MethodSignature methodSignature = (MethodSignature) joinPoint.getSignature();
    Method method = methodSignature.getMethod();
    return method.getAnnotation(clazz);
}
```

1. 在framework里面制定一个一个注解类

```
package com.my.blog.annotation;


import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;


@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface SystemLog {
    String businessName() default "";
}
```

2. 添加日志增强类，按要求打印信息

```
package com.my.blog.aspect;

@Component
@Aspect
@Slf4j
public class LogAspect {

    @Pointcut("@annotation(com.my.blog.annotation.SystemLog)")
    public void pointCut() {
    }

    @Around("pointCut()")
    public Object printLog(ProceedingJoinPoint joinPoint) throws Throwable {

        Object ret;
        try {
            handleBefore(joinPoint);
            ret = joinPoint.proceed();
            handleAfter(ret);
        } finally {
            // 结束后换行
            log.info("=======End=======" + System.lineSeparator());
        }
        return ret;
    }

    private void handleBefore(ProceedingJoinPoint joinPoint) {
        //获取本次请求的request
        ServletRequestAttributes requestAttributes = (ServletRequestAttributes)
RequestContextHolder.getRequestAttributes();

        if (requestAttributes != null) {
            MethodSignature signature = (MethodSignature)
joinPoint.getSignature();
            String businessName =
signature.getMethod().getAnnotation(SystemLog.class).businessName();
            log.info("=============Start=============");
            // 打印请求 URL
            log.info("URL            : {}",
requestAttributes.getRequest().getRequestURL().toString());
            // 打印描述信息
            log.info("BusinessName   : {}", businessName);
            // 打印 Http method
            log.info("HTTP Method    : {}",
requestAttributes.getRequest().getMethod());
            // 打印调用 controller 的全路径以及执行方法
            log.info("Class Method   : {}.{}",
joinPoint.getSignature().getDeclaringTypeName(),
joinPoint.getSignature().getName());
            // 打印请求的 IP
            log.info("IP             : {}",
requestAttributes.getRequest().getRemoteAddr());
            // 打印请求入参
            log.info("Request Args   : {}", joinPoint.getArgs());
        }
```

```
    }

    private void handleAfter(Object ret) {
        // 打印出参
        log.info("Response Args  : {}", JSON.toJSONString(ret));
    }

}
```

3. 在UserController 的userInfo方法上加上自定义注解

```
    @PutMapping("/userInfo")
    @SystemLog(businessName = "更新用户信息接口")
    public ResponseResult updateUserInfo(@RequestBody User user){
        return userService.updateUserInfo(user);
    }
```

3.18 更新浏览次数

3.18.1 需求

在用户浏览博文时要实现对应博客浏览量的增加。

3.18.2 思路分析

我们只需要在每次用户浏览博客时更新对应的浏览数即可。

但是如果直接操作博客表的浏览量的话，在并发量大的情况下会出现什么问题呢?

如何去优化呢?

①在应用启动时把博客的浏览量存储到redis中

②更新浏览量时去更新redis中的数据

③每隔10分钟把Redis中的浏览量更新到数据库中

④读取文章浏览量时从redis读取

3.18.3 铺垫知识

3.18.3.1 CommandLineRunner实现项目启动时预处理

如果希望在SpringBoot应用启动时进行一些初始化操作可以选择使用CommandLineRunner来进行处理。

我们只需要实现CommandLineRunner接口，并且把对应的bean注入容器。把相关初始化的代码重新到需要重新的方法中。

这样就会在应用启动的时候执行对应的代码。

```java
@Component
public class TestRunner implements CommandLineRunner {
    @Override
    public void run(String... args) throws Exception {
        System.out.println("程序初始化");
    }
}
```