//配置单个文件上传大小

spring.sservlet.multipart.max-file-size=10M

//配置单次请求最大文件上传大小

spring.sservlet.multipart.max-request-size=10M

public class UploadController {  
 @Autowired  
 private AliOSSUtils aliOSSUtils;

//本地存储文件  
 @PostMapping("/upload")  
 public Result upload(String username , Integer age , MultipartFile image) throws Exception {  
 log.info("文件上传: {}, {}, {}", username, age, image);  
 //获取原始文件名 - 1.jpg 123.0.0.jpg  
 String originalFilename = image.getOriginalFilename();  
 //构造唯一的文件名 (不能重复) - uuid(通用唯一识别码) de49685b-61c0-4b11-80fa-c71e95924018

int index = originalFilename.lastIndexOf(".");  
 String extname = originalFilename.substring(index);  
 String newFileName = UUID.randomUUID().toString() + extname;  
 log.info("新的文件名: {}", newFileName);  
  
 //将文件存储在服务器的磁盘目录中 E:\images  
 image.transferTo(new File("E:\\images\\"+newFileName));  
  
 return Result.success();  
 }

@PostMapping("/upload")  
 public Result upload(MultipartFile image) throws IOException {  
 log.info("文件上传, 文件名: {}", image.getOriginalFilename());  
  
 //调用阿里云OSS工具类进行文件上传  
 String url = aliOSSUtils.upload(image);  
 log.info("文件上传完成,文件访问的url: {}", url);  
  
 return Result.success(url);  
 }  
  
}

AliOSSUtils

@Component  
public class AliOSSUtils { 用ConfigurationProperties等价替换  
// @Value("${aliyun.oss.endpoint}") @Data  
// private String endpoint ; @Component  
// @Value("${aliyun.oss.accessKeyId}") @ConfigurationProperties(prefix = "aliyun.oss")  
// private String accessKeyId ; public class AliOSSProperties {  
// @Value("${aliyun.oss.accessKeySecret}") private String endpoint;  
// private String accessKeySecret ; private String accessKeyId;  
// @Value("${aliyun.oss.bucketName}") private String accessKeySecret;  
// private String bucketName ; private String bucketName; }  
 有个依赖没加spring-boot-configuration-processor  
 @Autowired  
 private AliOSSProperties aliOSSProperties;  
 /\*\*  
 \* 实现上传图片到OSS  
 \*/  
 public String upload(MultipartFile file) throws IOException {  
 //获取阿里云OSS参数  
 String endpoint = aliOSSProperties.getEndpoint();  
 String accessKeyId = aliOSSProperties.getAccessKeyId();  
 String accessKeySecret = aliOSSProperties.getAccessKeySecret();  
 String bucketName = aliOSSProperties.getBucketName();  
  
 // 获取上传的文件的输入流  
 InputStream inputStream = file.getInputStream();  
  
 // 避免文件覆盖  
 String originalFilename = file.getOriginalFilename();  
 String fileName = UUID.randomUUID().toString() + originalFilename.substring(originalFilename.lastIndexOf("."));  
  
 //上传文件到 OSS  
 OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);  
 ossClient.putObject(bucketName, fileName, inputStream);  
  
 //文件访问路径  
 String url = endpoint.split("//")[0] + "//" + bucketName + "." + endpoint.split("//")[1] + "/" + fileName;  
 // 关闭ossClient  
 ossClient.shutdown();  
 return url;// 把上传到oss的路径返回  
 }  
}

JWT

第一部分：Header(头）， 记录令牌类型、签名算法等。 例如：{"alg":"HS256","type":"JWT"}

第二部分：Payload(有效载荷），携带一些自定义信息、默认信息等。 例如：{"id":"1","username":"Tom"}

第三部分：Signature(签名），防止Token被篡改、确保安全性。将header、payload，并加入指定秘钥，通过指定签名算法计算而来。

/\*\*  
 \* 生成JWT  
 \*/  
public void testGenJwt(){  
 Map<String, Object> claims = new HashMap<>();  
 claims.put("id",1);  
 claims.put("name","tom");  
  
 String jwt = Jwts.builder()  
 .signWith(SignatureAlgorithm.HS256, "itheima") //签名算法  
 .setClaims(claims) //自定义内容(载荷)  
 .setExpiration(new Date(System.currentTimeMillis() + 3600 \* 1000))//设置有效期为1h  
 .compact(); //获取令牌  
 System.out.println(jwt);  
}  
  
/\*\*  
 \* 解析JWT  
 \*/  
public void testParseJwt(){  
 Claims claims = Jwts.parser()  
 .setSigningKey("itheima")  
 .parseClaimsJws("eyJhbGciOiJIUzI1NiJ9.eyJuYW1lIjoidG9tIiwiaWQiOjEsImV4cCI6MTY3MDQ2NDU0N30.yPLRyiusrlrmWeC4-dhInjFuAghPkmiHSRHd\_DTKi9E")  
 .getBody();  
 System.out.println(claims);  
}

logincontroller

@PostMapping("/login")  
public Result login(@RequestBody Emp emp){  
 log.info("员工登录: {}", emp);  
 Emp e = empService.login(emp);  
  
 //登录成功,生成令牌,下发令牌  
 if (e != null){  
 Map<String, Object> claims = new HashMap<>();  
 claims.put("id", e.getId());  
 claims.put("name", e.getName());  
 claims.put("username", e.getUsername());  
  
 String jwt = JwtUtils.generateJwt(claims); //jwt包含了当前登录的员工信息  
 return Result.success(jwt);  
 }  
  
 //登录失败, 返回错误信息  
 return Result.error("用户名或密码错误");  
}

Filter

main函数上面@ServletComponentScan //开启了对servlet组件的支持

@WebFilter(urlPatterns = "/\*")  
public class DemoFilter implements Filter {  
 @Override //初始化方法, 只调用一次  
 public void init(FilterConfig filterConfig) throws ServletException {  
 System.out.println("init 初始化方法执行了");  
 }  
  
 @Override //拦截到请求之后调用, 调用多次  
 public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {  
 System.out.println("Demo 拦截到了请求...放行前逻辑");  
 //放行  
 chain.doFilter(request,response);  
 System.out.println("Demo 拦截到了请求...放行后逻辑");  
 }  
  
 @Override //销毁方法, 只调用一次  
 public void destroy() {  
 System.out.println("destroy 销毁方法执行了");  
 }

Loginfilter

@WebFilter(urlPatterns = "/\*") //设置拦截路径  
public class LoginCheckFilter implements Filter {  
 @Override  
 public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {  
 HttpServletRequest req = (HttpServletRequest) request;  
 HttpServletResponse resp = (HttpServletResponse) response;  
  
 //1.获取请求url。  
 String url = req.getRequestURL().toString();  
  
 //2.判断请求url中是否包含login，如果包含，说明是登录操作，放行  
 if(url.contains("login")){  
 chain.doFilter(request,response);  
 return;  
 }  
  
 //3.获取请求头中的令牌（token）。  
 String jwt = req.getHeader("token");  
  
 //4.判断令牌是否存在，如果不存在，返回错误结果（未登录）  
 if(!StringUtils.hasLength(jwt)){  
 Result error = Result.error("NOT\_LOGIN");  
 //手动转换 对象--json --------> 阿里巴巴fastJSON  
 String notLogin = JSONObject.toJSONString(error);  
 resp.getWriter().write(notLogin);  
 return;  
 }  
 //5.解析token，如果解析失败，返回错误结果（未登录）  
 try {  
 JwtUtils.parseJWT(jwt);  
 } catch (Exception e) {//jwt解析失败  
 e.printStackTrace();  
 Result error = Result.error("NOT\_LOGIN");  
 //手动转换 对象--json --------> 阿里巴巴fastJSON  
 String notLogin = JSONObject.toJSONString(error);  
 resp.getWriter().write(notLogin);  
 return;  
 }  
 //6.放行。  
 chain.doFilter(request, response);  
 }  
}

Interceptor

@Component  
public class LoginCheckInterceptor implements HandlerInterceptor {  
 @Override //目标资源方法运行前运行, 返回true: 放行, 放回false, 不放行  
 public boolean preHandle(HttpServletRequest req, HttpServletResponse resp, Object handler) throws Exception {  
 //1.获取请求url。  
 String url = req.getRequestURL().toString();  
 log.info("请求的url: {}",url);  
  
 //2.判断请求url中是否包含login，如果包含，说明是登录操作，放行。  
 if(url.contains("login")){  
 log.info("登录操作, 放行...");  
 return true;  
 }  
  
 //3.获取请求头中的令牌（token）。  
 String jwt = req.getHeader("token");  
  
 //4.判断令牌是否存在，如果不存在，返回错误结果（未登录）。  
 if(!StringUtils.hasLength(jwt)){  
 log.info("请求头token为空,返回未登录的信息");  
 Result error = Result.error("NOT\_LOGIN");  
 //手动转换 对象--json --------> 阿里巴巴fastJSON  
 String notLogin = JSONObject.toJSONString(error);  
 resp.getWriter().write(notLogin);  
 return false;  
 }  
  
 //5.解析token，如果解析失败，返回错误结果（未登录）。  
 try {  
 JwtUtils.parseJWT(jwt);  
 } catch (Exception e) {//jwt解析失败  
 e.printStackTrace();  
 log.info("解析令牌失败, 返回未登录错误信息");  
 Result error = Result.error("NOT\_LOGIN");  
 //手动转换 对象--json --------> 阿里巴巴fastJSON  
 String notLogin = JSONObject.toJSONString(error);  
 resp.getWriter().write(notLogin);  
 return false;  
 }  
  
 //6.放行。  
 log.info("令牌合法, 放行");  
 return true;  
 }  
  
 @Override //目标资源方法运行后运行  
 public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {  
 System.out.println("postHandle ...");  
 }  
  
 @Override //视图渲染完毕后运行, 最后运行  
 public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception {  
 System.out.println("afterCompletion...");  
 }  
}

还有一个配置类

@Configuration //配置类  
public class WebConfig implements WebMvcConfigurer {  
  
 @Autowired  
 private LoginCheckInterceptor loginCheckInterceptor;  
  
 @Override  
 public void addInterceptors(InterceptorRegistry registry) {  
 registry.addInterceptor(loginCheckInterceptor).addPathPatterns("/\*\*").excludePathPatterns("/login");  
 }  
}

全局异常处理类----RestControllerAdvice

@RestControllerAdvice  
public class GlobalExceptionHandler {  
  
 @ExceptionHandler(Exception.class)//捕获所有异常  
 public Result ex(Exception ex){  
 ex.printStackTrace();  
 return Result.error("对不起,操作失败,请联系管理员");  
 }  
  
}