链滴

spring security 集成 jwt

作者：wenyl

原文链接：https://ld246.com/article/1629095160242

来源网站：链滴

1、pom依赖引入

```
<dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-config</artifactId>
        <version>4.0.3.RELEASE</version>
    </dependency>
<dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-web</artifactId>
        <version>4.0.3.RELEASE</version>
    </dependency>
```

2、基础配置

properties文件

```
rsaPrivateKey=MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBAIxgS3b5J9IRY
vEFIEeDeQCGkOI5pT+Nl3wNe0fdWIiw36g4sH1l2sLuTZ6bew9YRLIapude6ORGZF5UfNy9cor3
7n3ew/fXCEKVRC6Kg+cREm1rqyMjDc9NtfksXG4RGf7GNeoTmUDVStsnXvoLnzvrE7FvbB12XK
hTQDnSZAgMBAAECgYBcyET45SP5x/2/87EtymSaAP3FB5aIgiIDIwMxsKpQa/PVHZfjZWVonn4
0QYYsFUaKhe0tXmEGiLRMWQGSkTEGfZ5l7uRmrNZ0Nk9asu4/fyjwZNHYDDGAELU5R4WgLv
09PdVLG/uyIxXh9qg9y9OpYM4KoATnsH7t7TPdI5gQJBAMnB6nz18BKSsHX7qDkWWpxZUOZ
cKIsZoaDlz1NkoAKrOuH8TYc75uwLhR17nOOI9kO+10FKlYQ/5+yUiQIE+kCQQCyHcU/0mlvvBy
JsYJMDUMtDk5/BtCWD6UZan/X1GH2EHx1W5LuL+wyIr6CUrY5G3osVU5ZvLly4zTFRQHr10xA
EAqFJT4zT7uUMQXR47VoVDyzTovY+xYFtSnd6jCs3w70n4wfeEUfUKIgV2LDPHYDixx6EsLIrmqy
7VGxdAxqKIQJAe2bOyvnfXK9KeXWCjMkeZ+/RGhBFXoC+0pdg4PHMDb7RaVgCc2nLPRKj3rlj
sNUNnvt3LgrnvOYXIHk/7IKcQJAGakaB+211CbKnIHtjxSsg07EiM3dZVA1jrXTbJ6NuhURZTIOR
YJsVdWoEbwBLv/STTvc7+G+wo1yuzzBAse+w==
rsaPublicKey=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCMYEt2+SfSEWJbxBSBHg
```

```
kAhpDiOaU/jZd8DXtH3ViIsN+oOLB9ZdrC7k2em3sPWESyGqbnXujkRmReVHzcvXKK9ze593sP
1whClUQuioPnERJta6sjIw3PTbX5LFxuERn+xjXqE5lA1UrbJ176C5876xOxb2wddlyn4U0A50mQI
AQAB
# AES密码加密私钥(Base64加密)
encryptAESKey=V2FuZzkyNjQ1NGRTQkFQSUpXVA==
# JWT认证加密私钥(Base64加密)
encryptJWTKey=U0JBUElKV1RkV2FuZzkyNjQ1NA==
# AccessToken过期时间-5分钟-5*60(秒为单位)
accessTokenExpireTime=300
# RefreshToken过期时间-30分钟-30*60(秒为单位)
refreshTokenExpireTime=1800
# Shiro缓存过期时间-5分钟-5*60(秒为单位)(一般设置与AccessToken过期时间一致)
shiroCacheExpireTime=10
# 文件缓存地址
fileCachePath=D:\\download\\kmair\\contract\\file_backup
loginUrl=/user/loginByUsernameAndPassword
```

读取配置类（@Data是lombok注解）

```
@Data
public class SysProperties {
    private String rsaPublicKey;
    private String rsaPrivateKey;
    private String encryptAESKey;
    private String encryptJWTKey;
    private int accessTokenExpireTime;
    private int refreshTokenExpireTime;
    private int shiroCacheExpireTime;
    private String fileCachePath;
    private String loginUrl;
}

@Data
@EnableAutoConfiguration
@Configuration
@PropertySource("classpath:config.properties")
public class SysConfig {
    @Value("${rsaPublicKey}")
    private String rsaPublicKey;
    @Value("${rsaPrivateKey}")
    private String rsaPrivateKey;
    @Value("${encryptAESKey}")
    private String encryptAESKey;
    @Value("${encryptJWTKey}")
    private String encryptJWTKey;
    @Value("${accessTokenExpireTime}")
    private int accessTokenExpireTime;
    @Value("${refreshTokenExpireTime}")
    private int refreshTokenExpireTime;
    @Value("${shiroCacheExpireTime}")
    private int shiroCacheExpireTime;
    @Value("${fileCachePath}")
    private String fileCachePath;
    @Value("${loginUrl}")
```

```
    private String loginUrl;
    @Bean(value = "sysProperties",name = "sysProperties")
    public SysProperties init(){
        SysProperties sysProperties = new SysProperties();
        sysProperties.setRsaPublicKey(rsaPublicKey);
        sysProperties.setRsaPrivateKey(rsaPrivateKey);
        sysProperties.setEncryptAESKey(this.encryptAESKey);
        sysProperties.setEncryptJWTKey(this.encryptJWTKey);
        sysProperties.setAccessTokenExpireTime(this.accessTokenExpireTime);
        sysProperties.setRefreshTokenExpireTime(this.refreshTokenExpireTime);
        sysProperties.setShiroCacheExpireTime(this.shiroCacheExpireTime);
        sysProperties.setFileCachePath(fileCachePath);
        sysProperties.setLoginUrl(loginUrl);
        return sysProperties;
    }
}
```

3、response工具类

这里有一个坑要注意避免，部分版本的fastjson在处理数据返回的时候，会调用response的getWrite法，所以我们这里需要调用getOutputstream方法，不然会出现报错，提示getWrite已经被调用过我的fastjson版本是1.2.70

```
/**
 * @author Mr.Wen
 * @version 1.0
 * @date 2021-08-11 15:34
 */
@Slf4j
public class ResponseUtils {
    public static void out(HttpServletResponse response, HttpResult result){
        response.setStatus(HttpStatus.OK.value());
        //统一返回的JSON数据
        response.setContentType("application/json; charset=UTF-8");
        try (ServletOutputStream out = response.getOutputStream()) {
            out.write(JSON.toJSONString(result).getBytes(StandardCharsets.UTF_8));
        } catch (IOException e) {
            log.error("输出数据异常", e);
        }
    }
}
```

4、常量类

```
package com.kmair.ky.contract.common.entity;


/**
 * 常量
 * @author dolyw.com
 * @date 2018/9/3 16:03
 */
public class Constant {
```

```
private Constant() {}

/**
 * redis-OK
 */
public static final String OK = "OK";

/**
 * redis过期时间，以秒为单位，一分钟
 */
public static final int EXRP_MINUTE = 60;

/**
 * redis过期时间，以秒为单位，一小时
 */
public static final int EXRP_HOUR = 60 * 60;

/**
 * redis过期时间，以秒为单位，一天
 */
public static final int EXRP_DAY = 60 * 60 * 24;

/**
 * redis-key-前缀-shiro:access_token:
 */
public static final String PREFIX_SHIRO_ACCESS_TOKEN = "shiro:access_token:";
/**
 * redis-key-前缀-shiro:refresh_token_prefix:
 */
public static final String PREFIX_SHIRO_USER = "shiro:user_prefix:";
/**
 * redis-key-前缀-shiro:refresh_token:
 */
public static final String PREFIX_SHIRO_REFRESH_TOKEN = "shiro:refresh_token:";
/**
 * redis-key-前缀-security:user_details:
 */
public static final String PREFIX_SECURITY_USER_DETAILS = "security:user_details:";
/**
 * JWT-account:
 */
public static final String ACCOUNT = "account";

/**
 * jwt-存储权限信息
 */
public static final String AUTHORITIES = "authorities";

/**
 * JWT-currentTimeMillis:
 */
public static final String CURRENT_TIME_MILLIS = "currentTimeMillis";
```

```
    /**
     * PASSWORD_MAX_LEN
     */
    public static final Integer PASSWORD_MAX_LEN = 8;


    /**
     * 资源目录菜单标识
     */
    public static final String MENU_CODE = "menu";

}
```

5、用户信息类重写

```
package com.kmair.ky.contract.config.security;

import lombok.Data;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

/**
 * @author Mr.Wen
 * @version 1.0
 * @date 2021-08-11 15:07
 */
@Data
public class SecurityUserDetails implements UserDetails {
    private String username;
    private String passwrod;
    /**
     * 权限标识集合
     */
    private List<String> permissions;
    public SecurityUserDetails() {}
    public SecurityUserDetails(String username, String passwrod, List<String> permissions) {
        this.username = username;
        this.passwrod = passwrod;
        this.permissions = permissions;
    }

    /**
     * 账号是否过期
     */
    private Boolean isAccountExpired = true;
    /**
     * 账号是否锁定
     */
```

```
    private Boolean isAccountLocked = true;
    /**
     * 密码是否过期
     */
    private Boolean isCredentialsExpired = true;
    /**
     * 是否被禁用
     */
    private Boolean isEnabled = true;
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        Collection<GrantedAuthority> authorities =new ArrayList<>();
        for (String permisson : permissions) {
            if(permisson!=null) {
                GrantedAuthority authority=new SimpleGrantedAuthority(permisson);
                authorities.add(authority);
            }
        }
        return authorities;
    }

    @Override
    public String getPassword() {
        return passwrod;
    }

    @Override
    public String getUsername() {
        return username;
    }

    @Override
    public boolean isAccountNonExpired() {
        return isAccountExpired;
    }

    @Override
    public boolean isAccountNonLocked() {
        return isAccountLocked;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return isCredentialsExpired;
    }

    @Override
    public boolean isEnabled() {
        return isEnabled;
    }
}
```

6、用户信息查询类重写

这里要实现查询用户信息，关键就是实现UserDetailsService接口，内部实现，根据自己的需求来写

```
/**
 * @author Mr.Wen
 * @version 1.0
 * @date 2021-08-11 15:05
 */
@Service
public class UserDetailsServiceImpl implements UserDetailsService {
    @Resource
    private UserDataMapper userDataMapper;
    @Resource
    private SysRoleMapper sysRoleMapper;
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundExce
tion {
        UserData userData = userDataMapper.selectByWorkId(username);
        List<SysRole> sysRoles = sysRoleMapper.selectByUserId(userData.getId());
        List<String> permissions = new ArrayList<>();
        for(SysRole sysRole:sysRoles){
            permissions.add(sysRole.getRoleCode());
        }
        return new SecurityUserDetails(userData.getUsername(),userData.getPassword(),permissi
ns);
    }
}
```

7、授权过滤器，处理token

我使用了双token，这里也是根据自己的认真逻辑修改就好了

```
package com.kmair.ky.contract.config.security;

import com.kmair.ky.contract.common.api.HttpResult;
import com.kmair.ky.contract.common.entity.Constant;
import com.kmair.ky.contract.config.common.SysProperties;
import com.kmair.ky.contract.utils.auth.JwtUtil;
import com.kmair.ky.contract.utils.cache.JedisUtil;
import org.apache.commons.lang3.StringUtils;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.annotation.Resource;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.List;

/**
 * @author Mr.Wen
```

```
 * @version 1.0
 * @date 2021-08-11 15:20
 */
@Component
public class SecurityAuthorizeFilter extends OncePerRequestFilter {
    @Resource
    private UserDetailsServiceImpl userDetailsService;
    @Resource
    private SysProperties sysProperties;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response, FilterChain chain)
            throws IOException, ServletException {
        UsernamePasswordAuthenticationToken authentication=getAuthentication(request, res
onse);
        if(authentication!=null){
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
        chain.doFilter(request, response);
    }

    private UsernamePasswordAuthenticationToken getAuthentication(HttpServletRequest req
est,HttpServletResponse response) {
        String currentUrl = request.getServletPath();
        if(currentUrl.equals(sysProperties.getLoginUrl())){
            return null;
        }
        //查看请求头是否有 token
        String token = request.getHeader("Authorization");
        if(!StringUtils.isEmpty(token)){
            String username = null;
            List<String> permissions = null;
            try {
                username = JwtUtil.getClaim(token, Constant.ACCOUNT);
            } catch (Exception e) {
                logger.error("token解析异常",e);
                ResponseUtils.out(response, HttpResult.customCode(4012,"token为空，请登录",null))

                return null;
            }
            // refreshToken过期,提示重新登陆
            if(!JedisUtil.exists(Constant.PREFIX_SHIRO_REFRESH_TOKEN+username)){
                ResponseUtils.out(response, HttpResult.customCode(4011,"token过期，请登录",null))

                return null;
            }
            //accessToken过期，refreshToken未过期，重新签发token
            if(!JedisUtil.exists(Constant.PREFIX_SHIRO_ACCESS_TOKEN+username)){
                SecurityUserDetails userDetails = null;
                if(!JedisUtil.exists(Constant.PREFIX_SECURITY_USER_DETAILS + username)){
                    userDetails = (SecurityUserDetails) userDetailsService.loadUserByUsername(usern
me);
                }else{
```

```
                userDetails = (SecurityUserDetails)JedisUtil.getObject(Constant.PREFIX_SECURITY
USER_DETAILS + username);
            }
            // 拿到refreshToken签发的日期
            String refreshToken = request.getHeader("refreshToken");
            try {
                String timeStr = JwtUtil.getClaim(refreshToken, Constant.CURRENT_TIME_MILLIS);
                long time = Long.valueOf(timeStr);
                long currentTimeMillis = System.currentTimeMillis();
                String newAccessToken = JwtUtil.sign(username,String.valueOf(currentTimeMillis)
;
                // 在使用期间，新的accessToken时间大于refreshToke时间，表示用户正在使用，且ref
eshToken即将过期，两个token都要重新签发
                // 利用currentTimeMillis和token的时间，也可以判断用户是否很久没有操作了
                if((currentTimeMillis + sysProperties.getAccessTokenExpireTime()) > (time+sysPr
operties.getRefreshTokenExpireTime())){
                    JedisUtil.delKey(Constant.PREFIX_SECURITY_USER_DETAILS + username);
                    JedisUtil.delKey(Constant.PREFIX_SHIRO_REFRESH_TOKEN+username);
                    String newRefreshToken = JwtUtil.sign(username,String.valueOf(currentTimeMi
lis));
                    JedisUtil.setObject(Constant.PREFIX_SHIRO_REFRESH_TOKEN + username, new
efreshToken, sysProperties.getRefreshTokenExpireTime());
                    JedisUtil.setObject(Constant.PREFIX_SECURITY_USER_DETAILS + username, use
Details, sysProperties.getRefreshTokenExpireTime());
                    response.setHeader("NWE_REFRESH_TOKEN",newRefreshToken);
                }
                JedisUtil.setObject(Constant.PREFIX_SHIRO_ACCESS_TOKEN + username,newAcc
ssToken,sysProperties.getAccessTokenExpireTime());
                response.setHeader("NWE_ACCESS_TOKEN",newAccessToken);
                return new UsernamePasswordAuthenticationToken(userDetails, null, null);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return null;
    }else{
        ResponseUtils.out(response, HttpResult.customCode(4012,"token为空，请登录",null));
        return null;
    }
  }
}
```

8、密码处理器

```
package com.kmair.ky.contract.config.security;

import com.kmair.ky.contract.config.common.SysProperties;
import com.kmair.ky.contract.utils.security.PBECoder;
import com.kmair.ky.contract.utils.security.RsaEncrypt;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Component;
```

```
import javax.annotation.Resource;

/**
 * spring security 密码处理的实现
 * @author Mr.Wen
 * @version 1.0
 * @date 2021-08-12 13:51
 */
@Component
@Slf4j
public class SecurityPasswordEncoder implements PasswordEncoder {
    @Override
    public String encode(CharSequence rawPassword){
        if (rawPassword == null) {
            throw new IllegalArgumentException("rawPassword cannot be null");
        }
        //将前端加密传输的密码解密，再使用PBE加密，和数据库中存储的密码做对比
        String result;
        try {
            result = PBECoder.encrypt(rawPassword.toString()).replaceAll("[\r|\n]","");
        } catch (Exception e) {
            log.error("用户密码解密出现异常",e);
            throw new RuntimeException("对用户输入的密码进行rsa解密出现异常",e);
        }
        return result;
    }

    @Override
    public boolean matches(CharSequence rawPassword, String encodedPassword) {
        if (rawPassword == null) {
            throw new IllegalArgumentException("rawPassword cannot be null");
        }
        if (encodedPassword == null || encodedPassword.length() == 0) {
            return false;
        }
        return encode(rawPassword).equals(encodedPassword);
    }
}
```

9、security运行配置

```
package com.kmair.ky.contract.config.security;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.builders.Authenticatio
ManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.builders.WebSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfig
rerAdapter;
```

```
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFil
er;

import javax.annotation.Resource;

/**
 * @author Mr.Wen
 * @version 1.0
 * @date 2021-08-11 15:16
 */
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Resource
    private UserDetailsService userDetailsService;
    @Resource
    private SecurityAuthorizeFilter customAuthorizeFilter;
    @Resource
    private SecurityPasswordEncoder securityPasswordEncoder;
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            //必须放在UsernamePasswordAuthenticationFilter之前，如果请求头中替代有Token 就
接放行,没有的话 再进行用户名密码登陆
            .addFilterBefore(customAuthorizeFilter, UsernamePasswordAuthenticationFilter.class)
            //异常处理器
            .exceptionHandling()
            .and()
            // 授权请求
            .authorizeRequests()
            //permitAll 不需要任何授权 就可以访问
            //authenticated 需要登陆了 才能进行访问
//            .antMatchers("/**").authenticated()
            .antMatchers("/user/loginByUsernameAndPassword","/user/refreshToken/","/user/get
serInfoByToken").permitAll()
            .and()
            //修改session策略 无状态应用 STATELESS 因为没用到session
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            //解决跨域访问
            .cors().and().csrf().disable();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(securityPasswordEncoder);
    }

    @Override
    public void configure(WebSecurity web){
        web.ignoring().antMatchers("/swagger-ui.html", "/swagger-resources/**", "/v2/api-docs",
```

```
"/v2/**", "/webjars/**");
    }

    @Bean
    public AuthenticationManager getAuthenticationManager() throws Exception{
        return super.authenticationManager();
    }
}
```

10、登录，退出接口

这个类的login方法涉及到一个加密解密的算法，前端传递的用户密码信息都是用这个算法加密解密，换成自己的实现即可；根据用户令牌获取用户权限的接口自己实现就好；退出没有调用spring security的方法，自己实现了一个，另一种方法就是使用默认的退出登录的接口，然后自己实现一个LogoutuccessHandler，在这里面处理退出的逻辑即可

```
package com.kmair.ky.contract.system.login.controller;

import com.kmair.ky.contract.common.api.HttpResult;
import com.kmair.ky.contract.common.entity.Constant;
import com.kmair.ky.contract.config.common.SysProperties;
import com.kmair.ky.contract.config.security.SecurityUserDetails;
import com.kmair.ky.contract.system.login.service.impl.LoginServiceImpl;
import com.kmair.ky.contract.system.user.entity.UserData;
import com.kmair.ky.contract.utils.auth.JwtUtil;
import com.kmair.ky.contract.utils.cache.JedisUtil;
import com.kmair.ky.contract.utils.security.RsaEncrypt;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiParam;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContext;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.*;

import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.UnsupportedEncodingException;
import java.security.spec.InvalidKeySpecException;
import java.util.HashMap;
import java.util.Map;

/**
 * 登录
 * @author Mr.Wen
 */
@RestController
```

```
@RequestMapping("/user")
@Api(tags = "用户登录接口")
@Slf4j
public class LoginController {
    @Resource
    private SysProperties sysProperties;
    @Resource
    private LoginServiceImpl loginService;
    @Resource
    private RsaEncrypt rsaEncrypt;
    @Resource
    private AuthenticationManager authenticationManager;

    @RequestMapping(value = "/loginPage",method = RequestMethod.GET)
    @ApiOperation("进入登录页面")
    public String loginPage() {
        return "login page";
    }

    /**
     * 通过用户名密码登录
     * @param user 用户信息
     * @param response 请求的响应
     * @return 返回登录状态
     */
    @PostMapping("/loginByUsernameAndPassword")
    @ApiOperation("使用用户名登录")
    public HttpResult loginByUsernameAndPassword(@RequestBody @ApiParam(name="user"
value="用户信息",required=true) UserData user, HttpServletResponse response) {
        try {
            // 查询数据库中的帐号信息
            String username = rsaEncrypt.decrypt(user.getUsername(),sysProperties.getRsaPrivate
ey());
            String password = rsaEncrypt.decrypt(user.getPassword(),sysProperties.getRsaPrivateK
y());
            // 清空redis
            JedisUtil.delKey(Constant.PREFIX_SHIRO_ACCESS_TOKEN + username);
            JedisUtil.delKey(Constant.PREFIX_SHIRO_REFRESH_TOKEN + username);
            JedisUtil.delKey(Constant.PREFIX_SECURITY_USER_DETAILS + username);
            UsernamePasswordAuthenticationToken authRequest =
                    new UsernamePasswordAuthenticationToken(username, password);
            Authentication authentication = authenticationManager.authenticate(authRequest);
            SecurityContextHolder.getContext().setAuthentication(authentication);
            SecurityUserDetails userDetails = (SecurityUserDetails) authentication.getPrincipal();
            //存储token
            String currentTime = String .valueOf(System.currentTimeMillis());
            String accessToken = JwtUtil.sign(userDetails.getUsername(),currentTime);
            String refreshToken = JwtUtil.sign(userDetails.getUsername(),currentTime);
            JedisUtil.setObject(Constant.PREFIX_SHIRO_ACCESS_TOKEN + username,accessToken,
ysProperties.getAccessTokenExpireTime());
            JedisUtil.setObject(Constant.PREFIX_SHIRO_REFRESH_TOKEN + username, refreshTok
n, sysProperties.getRefreshTokenExpireTime());
            JedisUtil.setObject(Constant.PREFIX_SECURITY_USER_DETAILS + username, userDetails,
sysProperties.getRefreshTokenExpireTime());
```

```
            Map<String,String> ret = new HashMap<>(4);
            ret.put("accessToken",accessToken);
            ret.put("refreshToken",refreshToken);
            response.setHeader("Access-Control-Expose-Headers", "Authorization");
            return HttpResult.success("登陆成功",ret);
        }catch (Exception e){
            e.printStackTrace();
            if(e instanceof BadCredentialsException){
                return HttpResult.error("登陆失败--密码错误");
            } else if (e instanceof UnsupportedEncodingException){
                return HttpResult.error("登陆失败--不支持的编码");
            }else if(e instanceof InvalidKeySpecException){
                return HttpResult.error("登陆失败--rsa的key不合法");
            }else{
                return HttpResult.error("登陆失败，请联系管理员");
            }
        }
    }

    /**
     * 通过token获取用户信息
     * @param request 请求体
     * @return 用户信息（用户名，角色，资源权限）
     */
    @GetMapping("/getUserInfoByToken")
    @ApiOperation("根据token查询用户权限信息")
    public HttpResult getUserInfoByToken(HttpServletRequest request) throws Exception{
        String authorization = request.getHeader("Authorization");
        // 解密获得Account
        String username = JwtUtil.getClaim(authorization, Constant.ACCOUNT);
        Map<String, Object> userInfo = loginService.getUserInfoByUsername(username);
        return HttpResult.success(userInfo);
    }

    /**
     * 退出登录
     * @param request 请求体
     * @return 操作状态
     */
    @PostMapping("/logout")
    @ApiOperation("退出登录")
    public HttpResult logout(HttpServletRequest request)  throws Exception{
        String authorization = request.getHeader("Authorization");
        // 解密获得Account
        String username = JwtUtil.getClaim(authorization, Constant.ACCOUNT);
        Boolean accessTokenExist = JedisUtil.exists(Constant.PREFIX_SHIRO_ACCESS_TOKEN + u
ername);
        if (accessTokenExist != null && accessTokenExist) {
            JedisUtil.delKey(Constant.PREFIX_SHIRO_ACCESS_TOKEN + username);
        }
        Boolean refreshTokenExist = JedisUtil.exists(Constant.PREFIX_SHIRO_REFRESH_TOKEN +
sername);
        if (refreshTokenExist != null && refreshTokenExist) {
            JedisUtil.delKey(Constant.PREFIX_SHIRO_REFRESH_TOKEN + username);
```

```
        }
        Boolean userDetailsExist = JedisUtil.exists(Constant.PREFIX_SECURITY_USER_DETAILS + u
ername);
        if (userDetailsExist != null && userDetailsExist) {
            JedisUtil.delKey(Constant.PREFIX_SECURITY_USER_DETAILS + username);
        }
        HttpSession session = request.getSession(false);
        if (session != null) {
            session.invalidate();
        }
        SecurityContext context = SecurityContextHolder.getContext();
        context.setAuthentication(null);
        SecurityContextHolder.clearContext();
        return HttpResult.success("退出成功");
    }


}
```

11、总结

spring security可以自定义loginSuccessHandler，logoutSuccessHandler还有对应的失败的以及他的handler,总的来说功能还是比较强大，和shiro相比，二者能实现功能都差不多，但是spring security依赖于spring容器，而且spring security对新手不太友好，总的来说，可以用shiro就没必要使用spring security。