1.视频参考

https://www.bilibili.com/video/BV1mm4y1X7Hc?p=28&spm_id_from=pageDriver

(14条消息) SpringSecurity权限管理系统实战—九、数据权限的配置*CoderMy*的博客-*CSDN*博客springsecurity 数据权限

2.权限相关概念

认证：验证当前访问系统的是不是本系统的用户，并且要确认具体是哪个用户

授权：经过认证后判断打那个前用户是否有权限进行某个操作,不同用户不同权限

JWT

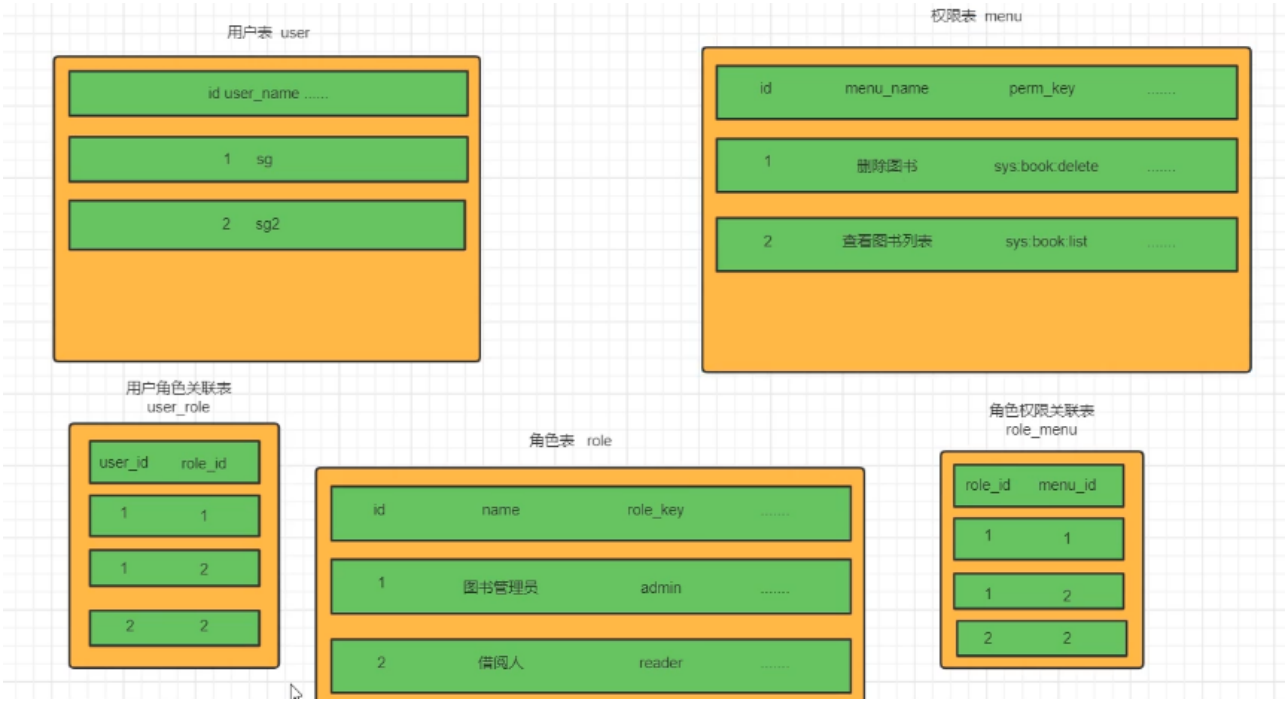
《若依分离版 V3.8.1》SpringSecurity 权限框架相关需实现或继承的类参考说明结合钟老师 pdf 文档对比 Shiro 理解：

相关类（序号不绝对顺序关系，理解）	说明
1、继承 WebSecurityConfigurerAdapter 抽象类	核心配置类
2、UsernamePasswordAuthenticationToken	实例化用户密码登录认证 token 对象
3、实现 UserDetailsService 接口	AuthenticationManager 对象执行 authenticate（传入 UsernamePasswordAuthenticationToken 对象）然后会执行 loadUserByUsername（）返回 Authentication 对象，里会边有 UserDetails 对象，若依这步会查找当前用户权限（进行用户名、密码校验,对比理解 Shiro 的自定义域 Realm）
4、LoginUser 实现 UserDetails 接口	理解为 Security 维护的用户对象，当前账户存储在 SecurityContextHolder 中，其他过滤器从这获取。
5、new BCryptPasswordEncoder()	加密，配置类 BCryptPasswordEncoder @Bean，强散列哈希算法实现
6、(LoginUser) authentication.getPrincipal()	从 Authentication 对象中拿到用户(校长)对象
7、生成 token	JWT 生成 token，Redis 存用户信息然后返回前端
8、UsernamePasswordAuthenticationFilter、OncePerRequestFilter 接口	自定义过滤器，后续请求先走这验证 token 校验(从 Redis 查找，存在则检验 token，更新 Authentication 对象)再放行。 Ps：OncePerRequestFilter 保证每次请求时只执行一次过滤。
9、授权	@PreAuthorize("@ss.hasPermi('system:post:list')") 结合认证自行授代码

@J8Studio

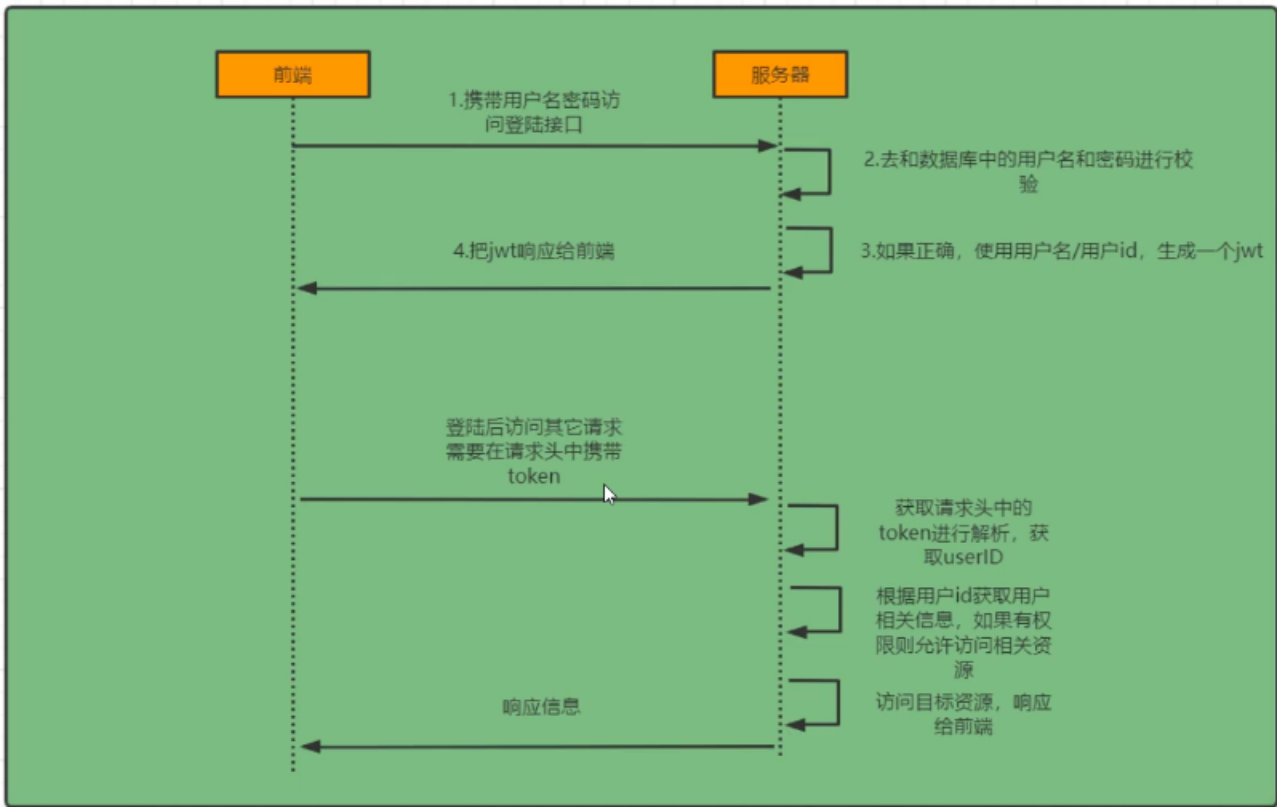
3.权限控制**5**表的**RBAC**模型

RBAC是Role Based Access Control的缩写，是基于角色的访问控制。一般都是分为用户（user），角色（role），权限（permission）三个实体，用户（user）和角色（role）是多对多的关系，角色（role）和权限（permission）也是多对多的 关系。用户（user）和权限（permission） 之间没有直接的关系，都是通过角色作为代理，才能获取到用户（user）拥有的权限。



4.Spring Security认证流程

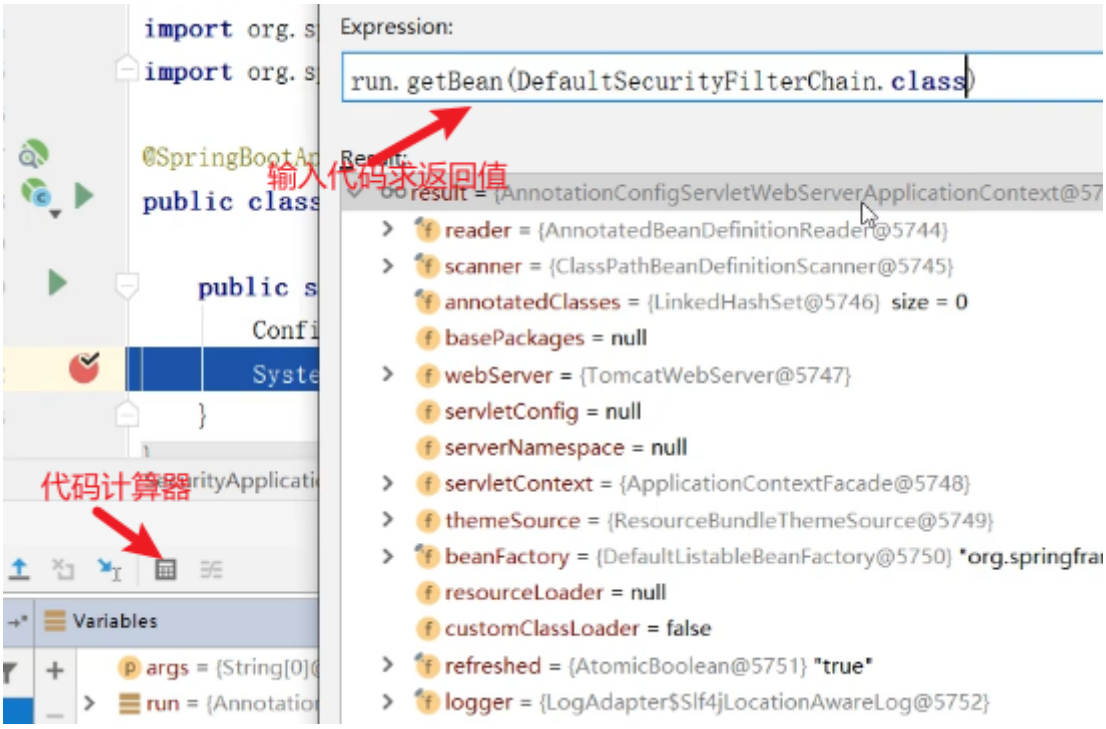
认证流程：

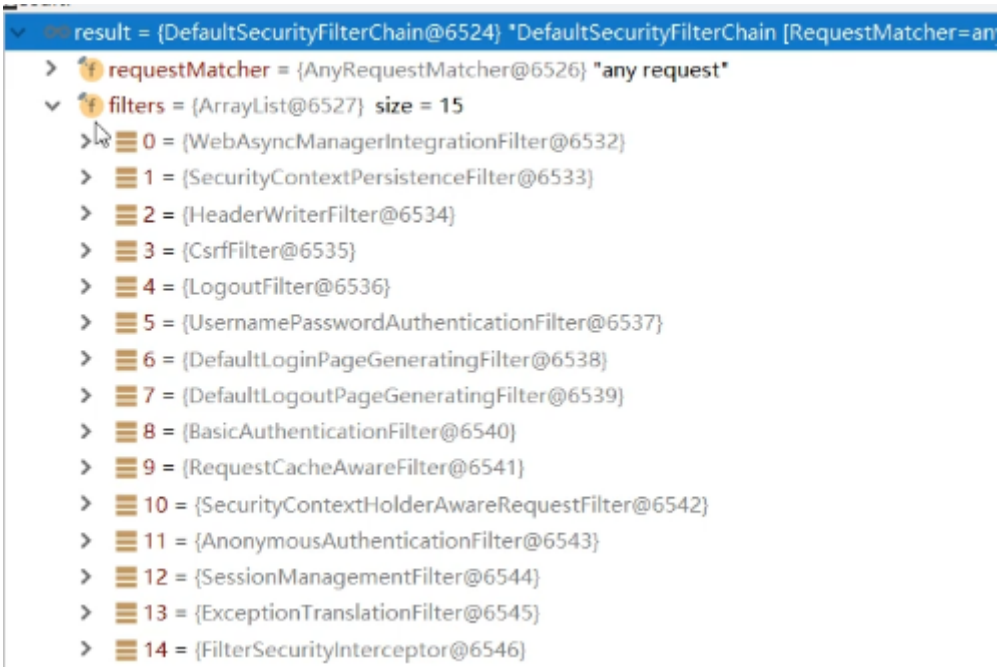


引用之后：1.有默认登录页2.默认登录账号密码3.session存储token4.从seesion中认证

Spring Security是基于过滤器链实现，修改上述4点实现自定义JWT认证：

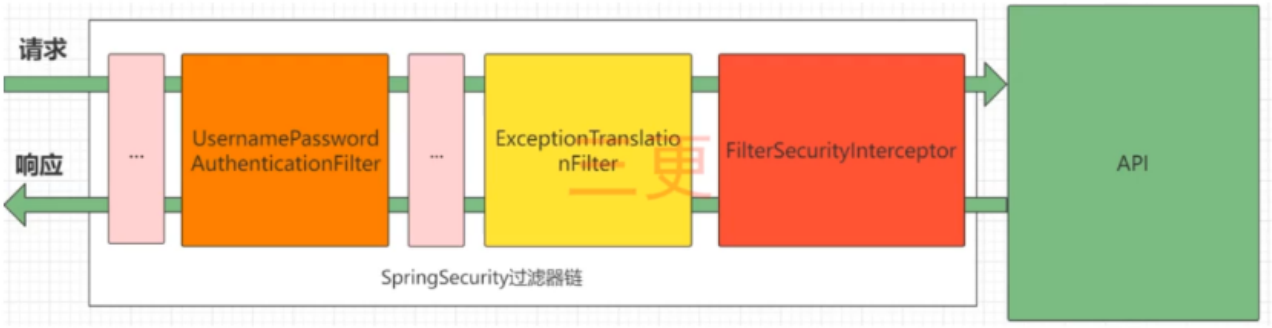
4.1可以在全局容器中取得过滤器链





4.2常用过滤器及其作用

SpringSecurity的原理其实就是一个过滤器链，内部包含了提供各种功能的过滤器。这里我们可以看看入门案例中的过滤器。



图中只展示了核心过滤器，其它的非核心过滤器并没有在图中展示。

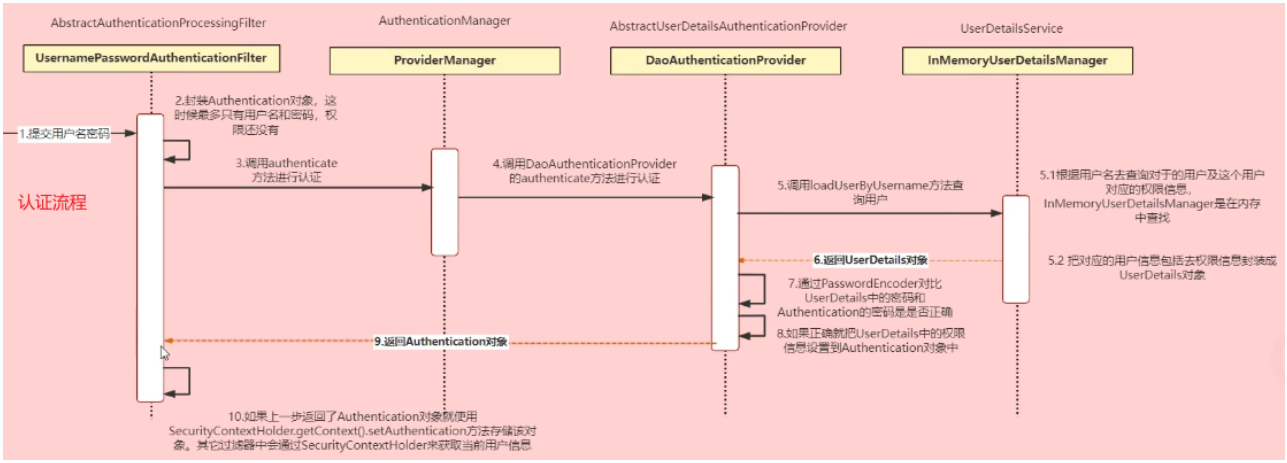
UsernamePasswordAuthenticationFilter:负责处理我们在登陆页面填写了用户名密码后的登陆请求。入门案例的认证工作主要有它负责。

ExceptionTranslationFilter：处理过滤器链中抛出的任何AccessDeniedException和AuthenticationException 。

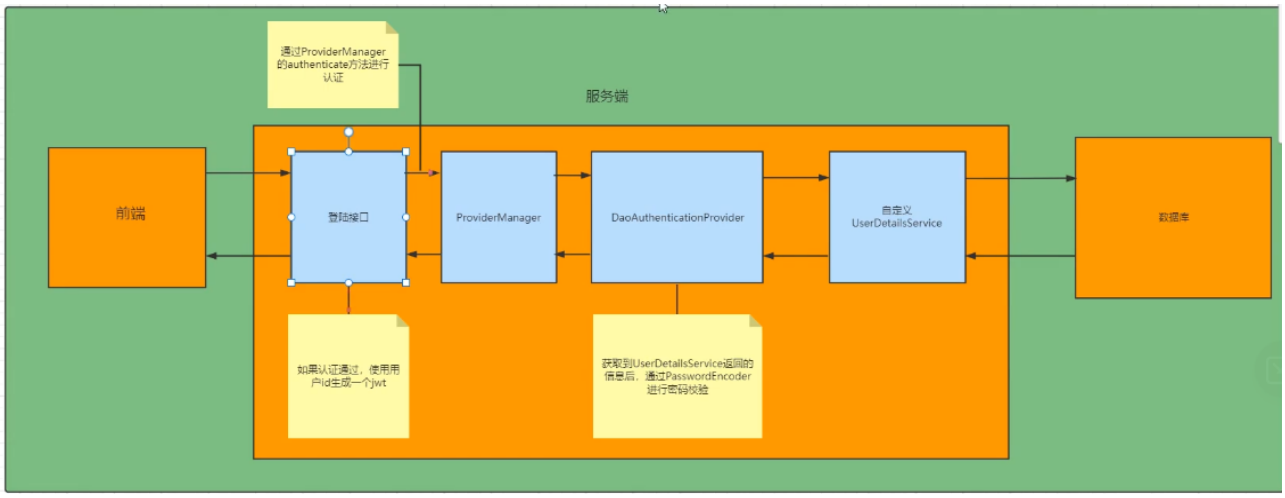
FilterSecurityInterceptor：负责权限校验的过滤器。

4.3认证过滤器UsernamePasswordAuthenticationFilter实现认证

引入reids依赖，jwt依赖，json转换工具依赖，各种工具类：JTW的，json的，redis的...通用配置即可

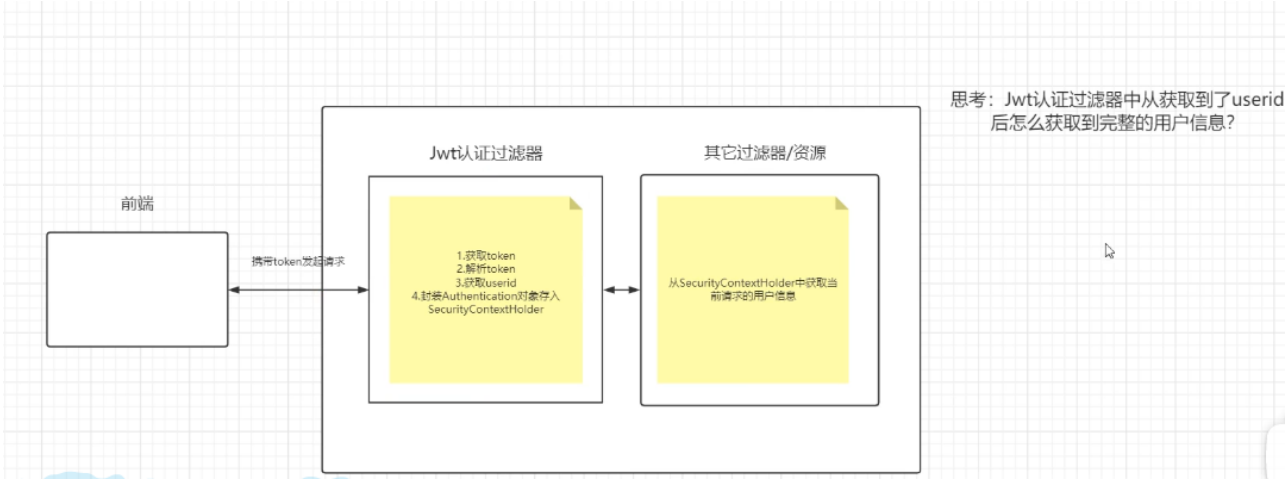


分析：1.自定义controller接口替代第一个filter，接口中在调用第二个filter自动调用到第五步　2.自定义5中的查找方式，去查找数据库认证　3.重写10，生成Token返回　4.最终根据用户ID将用户信息放在redis中



分析：后续请求携带Token，怎么校验过其他过滤器呢？

JTW过滤器：解析Token获得id，redis中获得用户信息封装成Auth对象封装进Hoder让后续过滤器调用



总结：

登录

①自定义登录接口

调用ProviderManager的方法进行认证 如果认证通过生成jwt

把用户信息存入redis中

②自定义UserDetailsService

在这个实现列中去查询数据库

校验：

①定义Jwt认证过滤器

获取token

解析token获取其中的userid

从redis中获取用户信息

存入SecurityContextHolder

4.6.若依实现

4.6.1UserDetails接口，自定义UserDetails对象

```
package com.ruoyi.common.core.domain.model;

import java.util.Collection;
import java.util.Set;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import com.alibaba.fastjson.annotation.JSONField;
import com.ruoyi.common.core.domain.entity.SysUser;

/**
 * 登录用户身份权限
 *
 * @author ruoyi
 */
public class LoginUser implements UserDetails
{
    private static final long serialVersionUID = 1L;

    /**
     * 用户ID
     */
```

```
    private Long userId;

    /**
     * 部门ID
     */
    private Long deptId;

    /**
     * 用户唯一标识
     */
    private String token;

    /**
     * 登录时间
     */
    private Long loginTime;

    /**
     * 过期时间
     */
    private Long expireTime;

    /**
     * 登录IP地址
     */
    private String ipaddr;

    /**
     * 登录地点
     */
    private String loginLocation;

    /**
     * 浏览器类型
     */
    private String browser;

    /**
     * 操作系统
     */
    private String os;

    /**
```

```
     * 权限列表
     */
    private Set<String> permissions;

    /**
     * 用户信息
     */
    private SysUser user;

    public Long getUserId()
    {
        return userId;
    }

    public void setUserId(Long userId)
    {
        this.userId = userId;
    }

    public Long getDeptId()
    {
        return deptId;
    }

    public void setDeptId(Long deptId)
    {
        this.deptId = deptId;
    }

    public String getToken()
    {
        return token;
    }

    public void setToken(String token)
    {
        this.token = token;
    }

    public LoginUser()
    {
    }
```

```
    public LoginUser(SysUser user, Set<String> permissions)
    {
        this.user = user;
        this.permissions = permissions;
    }

    public LoginUser(Long userId, Long deptId, SysUser user,
Set<String> permissions)
    {
        this.userId = userId;
        this.deptId = deptId;
        this.user = user;
        this.permissions = permissions;
    }

    @JSONField(serialize = false)
    @Override
    public String getPassword()
    {
        return user.getPassword();
    }

    @Override
    public String getUsername()
    {
        return user.getUserName();
    }

    /**
     * 账户是否未过期,过期无法验证
     */
    @JSONField(serialize = false)
    @Override
    public boolean isAccountNonExpired()
    {
        return true;
    }

    /**
     * 指定用户是否解锁,锁定的用户无法进行身份验证
     *
     * @return
     */
```

```
    @JSONField(serialize = false)
    @Override
    public boolean isAccountNonLocked()
    {
        return true;
    }

    /**
     * 指示是否已过期的用户的凭据(密码),过期的凭据防止认证
     *
     * @return
     */
    @JSONField(serialize = false)
    @Override
    public boolean isCredentialsNonExpired()
    {
        return true;
    }

    /**
     * 是否可用 ,禁用的用户不能身份验证
     *
     * @return
     */
    @JSONField(serialize = false)
    @Override
    public boolean isEnabled()
    {
        return true;
    }

    public Long getLoginTime()
    {
        return loginTime;
    }

    public void setLoginTime(Long loginTime)
    {
        this.loginTime = loginTime;
    }

    public String getIpaddr()
    {
```

```
        return ipaddr;
    }

    public void setIpaddr(String ipaddr)
    {
        this.ipaddr = ipaddr;
    }

    public String getLoginLocation()
    {
        return loginLocation;
    }

    public void setLoginLocation(String loginLocation)
    {
        this.loginLocation = loginLocation;
    }

    public String getBrowser()
    {
        return browser;
    }

    public void setBrowser(String browser)
    {
        this.browser = browser;
    }

    public String getOs()
    {
        return os;
    }

    public void setOs(String os)
    {
        this.os = os;
    }

    public Long getExpireTime()
    {
        return expireTime;
    }
```

```
    public void setExpireTime(Long expireTime)
    {
        this.expireTime = expireTime;
    }

    public Set<String> getPermissions()
    {
        return permissions;
    }

    public void setPermissions(Set<String> permissions)
    {
        this.permissions = permissions;
    }

    public SysUser getUser()
    {
        return user;
    }

    public void setUser(SysUser user)
    {
        this.user = user;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities()
    {
        return null;
    }
}
```

4.6.2.自定义UserDetailsService

```
package com.ruoyi.framework.web.service;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UserDetailsService;
```

```
import
org.springframework.security.core.userdetails.UsernameNotFoundExcep
tion;
import org.springframework.stereotype.Service;
import com.ruoyi.common.core.domain.entity.SysUser;
import com.ruoyi.common.core.domain.model.LoginUser;
import com.ruoyi.common.enums.UserStatus;
import com.ruoyi.common.exception.ServiceException;
import com.ruoyi.common.utils.StringUtils;
import com.ruoyi.system.service.ISysUserService;

/**
 * 用户验证处理
 *
 * @author ruoyi
 */
@Service
public class UserDetailsServiceImpl implements UserDetailsService
{
    private static final Logger log =
LoggerFactory.getLogger(UserDetailsServiceImpl.class);

    @Autowired
    private ISysUserService userService;

    @Autowired
    private SysPermissionService permissionService;

    //默认调用此方法查询密码
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException
    {
        SysUser user = userService.selectUserByUserName(username);
        if (StringUtils.isNull(user))
        {
            log.info("登录用户：{} 不存在.", username);
            throw new ServiceException("登录用户：" + username + " 不
存在");
        }
        else if
(UserStatus.DELETED.getCode().equals(user.getDelFlag()))
        {
```

```
            log.info("登录用户：{} 已被删除.", username);
            throw new ServiceException("对不起，您的账号：" + username
+ " 已被删除");
        }
        else if
(UserStatus.DISABLE.getCode().equals(user.getStatus()))
        {
            log.info("登录用户：{} 已被停用.", username);
            throw new ServiceException("对不起，您的账号：" + username
+ " 已停用");
        }

        return createLoginUser(user);
    }

    public UserDetails createLoginUser(SysUser user)
    {
        return new LoginUser(user.getUserId(), user.getDeptId(),
user, permissionService.getMenuPermission(user));
    }
}
```

4.6.3.SecurityConfig配置文件

```
    /**
     * 强散列哈希加密实现
     */
    @Bean
    public BCryptPasswordEncoder bCryptPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }

//原理：使用这两个方法进行密码加密与验证
String encode = new BCryptPasswordEncoder().encode("123");//密码加密
new BCryptPasswordEncoder().matches("1234",encode);//密码验证

//加密后：$2a$10$为盐，随机生成
$2a$10$7JB720yubVSZvUI0rEqK/.VqGOZTH.ulu33dHOiBE8ByOhJIrdAu2
//注册时把密码加密存储到数据库中，登录时比较
使用时去IOC容器中通过@Autowire拿到 @Bean时注册进去的对象
//Bean共享，那么如何保证Spring线程安全呢？
Spring 的常见业务组件采取单例如何保证线程安全
```

Spring 作为 IOC 框架，一般来说，Spring 管理的 controller、service、dao 都是单例存在，节省内存和 cpu、提高单机资源利用率（默认单例，配置多例模式使用 scope=prototype），既然是单例，那么如何控制单例被多个线程同时访问线程安全呢？

首先要理解每个 http 请求到后台都是一个单独的线程，线程之间共享同一个进程的内存、io、cpu 等资源，但线程栈是线程独有，线程之间不共享栈资源

其次，bean 分为有状态 bean 和无状态 bean，有状态 bean 即类定义了成员变量，可能被多个线程同时访问，则会出现线程安全问题；无状态 bean 每个线程访问不会产生线程安全问题，因为各个线程栈及方法栈资源都是独立的，不共享。即是，无状态 bean 可以在多线程环境下共享，有状态 bean 不能

Spring 的 dao、service 层使用的有状态 bean 如何保证线程安全？
Spring 应用中 dao、service 一般以单例形式存在，dao、service 中使用的数据库 connection 以及 RequestContextHolder、TransactionSynchronizationManager、LocaleContextHolder 等都是有状态 bean，而 dao、service 又是单例，如何保证线程安全呢？

答案是使用 threadLocal 进行处理，ThreadLocal 是线程本地变量，每个线程拥有变量的一个独立副本，所以各个线程之间互不影响，保证了线程安全

SpringMVC 的 controller 并发访问如何保证线程安全？
SpringMVC 中的 controller 默认是单例的，那么如果不小心在类中定义了类变量，那么这个类变量是被所有请求共享的，这可能会造成多个请求修改该变量的值，出现与预期结果不符合的异常。所以如上所述，属性变量会到值线程安全问题，解决方法包括使用 threadLocal 或不使用属性变量、配置为多例均可（加锁控制效率不行）

4.6.4.自定义登录接口

```
/**
     * 解决 无法直接注入 AuthenticationManager
     * 重写即可，login方法中使用AuthenticationManager
     * @return
     * @throws Exception
     */
    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws
Exception {
        return super.authenticationManagerBean();
    }
//登录放行设置
```

```
    @Override
    protected void configure(HttpSecurity httpSecurity) throws
Exception {
        httpSecurity
                // CSRF禁用，因为不使用session
                .csrf().disable()
                // 认证失败处理类

.exceptionHandling().authenticationEntryPoint(unauthorizedHandler).
and()
                // 基于token，所以不需要session

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.ST
ATELESS).and()
                // 过滤请求
                .authorizeRequests()
                // 对于登录login 注册register 验证码captchaImage 允许匿
名访问
                .antMatchers("/login", "/register",
"/captchaImage").anonymous()
                .antMatchers(
                        HttpMethod.GET,
                        "/",
                        "/*.html",
                        "/**/*.html",
                        "/**/*.css",
                        "/**/*.js",
                        "/profile/**"
                ).permitAll()
                .antMatchers("/swagger-ui.html").anonymous()
                .antMatchers("/swagger-resources/**").anonymous()
                .antMatchers("/webjars/**").anonymous()
                .antMatchers("/*/api-docs").anonymous()
                .antMatchers("/druid/**").anonymous()

                // 除上面外的所有请求全部需要鉴权认证
                .anyRequest().authenticated()
                .and()
                .headers().frameOptions().disable();
          //配置退出方法

 httpSecurity.logout().logoutUrl("/logout").logoutSuccessHandler(lo
goutSuccessHandler);
```

```
        // 添加JWT filter
        httpSecurity.addFilterBefore(authenticationTokenFilter,
UsernamePasswordAuthenticationFilter.class);
        // 添加CORS filter
        httpSecurity.addFilterBefore(corsFilter,
JwtAuthenticationTokenFilter.class);
        httpSecurity.addFilterBefore(corsFilter,
LogoutFilter.class);
    }
```

```
// 用户验证
            // 该方法会去调用UserDetailsServiceImpl.loadUserByUsername
            // 创建UserDetails实现类Loginuser
            authentication = authenticationManager
                    .authenticate(new
UsernamePasswordAuthenticationToken(username, password));
...
LoginUser loginUser = (LoginUser) authentication.getPrincipal();
```

4.6.5.定义JWT校验过滤器

```
package com.ruoyi.framework.security.filter;

import java.io.IOException;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.authentication.UsernamePasswordAuthent
icationToken;
import
org.springframework.security.core.context.SecurityContextHolder;
import
org.springframework.security.web.authentication.WebAuthenticationDe
tailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
import com.ruoyi.common.core.domain.model.LoginUser;
import com.ruoyi.common.utils.SecurityUtils;
```

```
import com.ruoyi.common.utils.StringUtils;
import com.ruoyi.framework.web.service.TokenService;

/**
 * token过滤器 验证token有效性
 *
 * @author ruoyi
 */
@Component
public class JwtAuthenticationTokenFilter extends
OncePerRequestFilter
{
    @Autowired
    private TokenService tokenService;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain chain)
            throws ServletException, IOException
    {
        LoginUser loginUser = tokenService.getLoginUser(request);
        if (StringUtils.isNotNull(loginUser) &&
StringUtils.isNull(SecurityUtils.getAuthentication()))
        {
            tokenService.verifyToken(loginUser);
            //必须使用三个参数的:   因为此构造中会设置认证状态
super.setAuthenticated(true);
            //1.principal 2.null    3.获取权限信息
            UsernamePasswordAuthenticationToken authenticationToken
= new UsernamePasswordAuthenticationToken(loginUser, null,
loginUser.getAuthorities());
            authenticationToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));

 SecurityContextHolder.getContext().setAuthentication(authenticatio
nToken);
        }
        //有没有token都放行：没有让后续过滤器抛异常
        chain.doFilter(request, response);
    }
}
```

4.6.6再配置

```
// 添加JWT filter到UsernamePasswordAuthenticationFilter之前
httpSecurity.addFilterBefore(authenticationTokenFilter,UsernamePass
wordAuthenticationFilter.class);
```

退出登录：删除redis用户数据

5.Spring Security授权流程

授权：

在SpringSecurity中，会使用默认的FilterSecurityInterceptor来进行权限校验。在FilterSecurityInterceptor中会从SecurityContextHolder获取其中的Authentication，然后获取其中的权限信息。当前用户是否拥有访问当前资源所需的权限。

所以我们在项目中只需要把当前登录用户的权限信息也存入Authentication。

然后设置我们的资源所需要的权限即可。

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
```

然后就可以使用对应的注解。@PreAuthorize

```
@RestController
public class HelloController {

    @RequestMapping("/hello")
    @PreAuthorize("hasAuthority('test')")
    public String hello(){
        return "hello";
    }
}
```

1.封装权限到userlogin中

```
public UserDetails createLoginUser(SysUser user)
    {
        return new LoginUser(user.getUserId(), user.getDeptId(),
user, permissionService.getMenuPermission(user));
    }
```

2.使用注解

```
  @PreAuthorize("@ss.hasPermi('system:role:list')")
//根据方法的返回值判断是否有权限。
@Service("ss")
```

```
public class PermissionService
{
    public boolean hasPermi(String permission)
    {
        if (StringUtils.isEmpty(permission))
        {
            return false;
        }
        LoginUser loginUser = SecurityUtils.getLoginUser();
        if (StringUtils.isNull(loginUser) ||
CollectionUtils.isEmpty(loginUser.getPermissions()))
        {
            return false;
        }
        return hasPermissions(loginUser.getPermissions(),
permission);
    }
}
```

3.filtersecitry过滤器会去loginuser中查找权限。