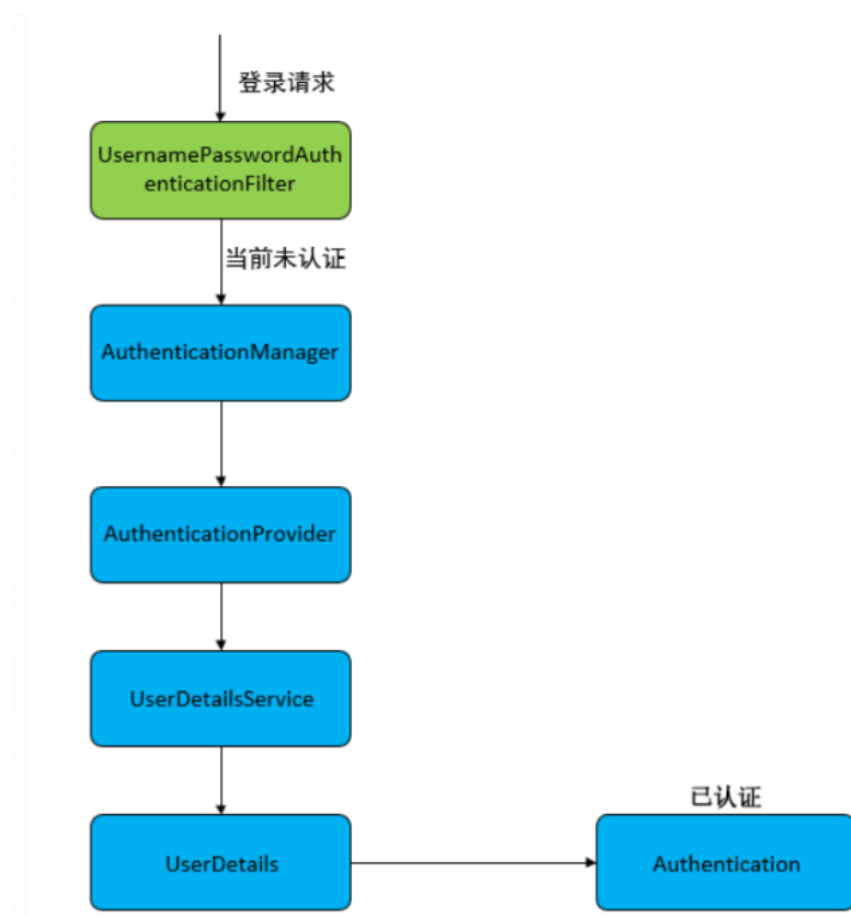


【第30话：很多人都回答不清楚的Spring Security认证流程】

Hello 小伙伴们，这节课给大家带来一个很多小伙伴都害怕的面试题：“Spring Security 认证流程”。

话不多说，我们先直接上图



下面我们跟下源码

1. 用户发起表单登录请求后，首先进入 UsernamePasswordAuthenticationFilter

```

public Authentication attemptAuthentication(HttpServletRequest request,
    HttpServletResponse response) throws AuthenticationException {
    if (postOnly && !request.getMethod().equals("POST")) {
        throw new AuthenticationServiceException(
            "Authentication method not supported: " + request.getMethod());
    }

    String username = obtainUsername(request);
    String password = obtainPassword(request);

    if (username == null) {
        username = "";
    }

    if (password == null) {
        password = "";
    }

    username = username.trim();

    UsernamePasswordAuthenticationToken authRequest = new UsernamePasswordAuthenticationToken(
        username, password);

    // Allow subclasses to set the "details" property
    setDetails(request, authRequest);

    return this.getAuthenticationManager().authenticate(authRequest);
}

public UsernamePasswordAuthenticationToken(Object principal, Object credentials) {
    super(authorities: null); // 因为此时还没有认证, 因此权限为 null
    this.principal = principal;
    this.credentials = credentials;
    setAuthenticated(false); // 没有认证, 设置为 false
}

```

在 UsernamePasswordAuthenticationFilter 中根据用户输入的用户名、密码构建了 UsernamePasswordAuthenticationToken，并将其交给 AuthenticationManager 来进行认证处理。

AuthenticationManager 本身不包含认证逻辑，其核心是用来管理所有的 AuthenticationProvider，通过交由合适的 AuthenticationProvider 来实现认证。

2.下面跳转到了 ProviderManager，该类是 AuthenticationManager 的实现类

```

public Authentication authenticate(Authentication authentication)
    throws AuthenticationException {
    Class<? extends Authentication> toTest = authentication.getClass();
    AuthenticationException lastException = null;
    Authentication result = null;
    boolean debug = logger.isDebugEnabled();

    for (AuthenticationProvider provider : getProviders()) {
        if (!provider.supports(toTest)) {
            continue;
        }

        if (debug) {
            logger.debug("Authentication attempt using "
                + provider.getClass().getName());
        }

        try {
            result = provider.authenticate(authentication);

            if (result != null) {
                copyDetails(authentication, result);
            }
        }
    }
}

```

我们知道不同的登录逻辑它的认证方式是不一样的，比如我们表单登录需要认证用户名和密码，但是当我们使用三方登录时就不需要验证密码。

Spring Security 支持多种认证逻辑，每一种认证逻辑的认证方式其实就是一种 AuthenticationProvider。通过 getProviders() 方法就能获取所有的 AuthenticationProvider，通过 provider.supports() 来判断 provider 是否支持当前的认证逻辑。当选择好一个合适的 AuthenticationProvider 后，通过 provider.authenticate(authentication) 来让 AuthenticationProvider 进行认证。

3.传统表单登录的 AuthenticationProvider 主要是由

AbstractUserDetailsAuthenticationProvider 来进行处理的，我们来看下它的 authenticate() 方法。

```
1 | user = retrieveUser(username, (UsernamePasswordAuthenticationToken) authentication);
```

retrieveUser() 的具体实现在 DaoAuthenticationProvider 中，代码如下：

```
protected final UserDetails retrieveUser(String username,
    UsernamePasswordAuthenticationToken authentication)
    throws AuthenticationException {
    prepareTimingAttackProtection();
    try {
        UserDetails loadedUser = this.getUserDetailsService().loadUserByUsername(username);
        if (loadedUser == null) {
            throw new InternalAuthenticationServiceException(
                "UserService returned null, which is an interface contract violation");
        }
        return loadedUser;
    } catch (UsernameNotFoundException ex) {
        mitigateAgainstTimingAttack(authentication);
        throw ex;
    } catch (InternalAuthenticationServiceException ex) {
        throw ex;
    } catch (Exception ex) {
        throw new InternalAuthenticationServiceException(ex.getMessage(), ex);
    }
}

@Override
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    Collection<GrantedAuthority> authorities = new ArrayList<>();
    // 从数据库中取出用户信息
    SysUser user = userService.selectByName(username);

    // 判断用户是否存在
    if (user == null) {
        throw new UsernameNotFoundException("用户名不存在");
    }

    // 添加权限
    List<SysUserRole> userRoles = userRoleService.listById(user.getId());
    for (SysUserRole userRole : userRoles) {
        SysRole role = roleService.selectById(userRole.getRoleId());
        authorities.add(new SimpleGrantedAuthority(role.getName()));
    }

    // 返回UserDetails实现类
    return new User(user.getName(), user.getPassword(), authorities);
}
```

getUserDetailsService(): 读取了框架中的 UserDetailsService。
在我们之前的程序中，使用了 CustomUserDetailsService 来实现 UserDetailsService，并将其注入到框架中。
这里就将我们自定义的 UserDetailsService 取出来，并通过 loadUserByUsername() 取出注入的 UserDetails。

当我们成功的读取 UserDetails 后，下面开始对其进行认证：

```
try {
    // 进行前认证校验
    preAuthenticationChecks.check(user);
    additionalAuthenticationChecks(user,
        (UsernamePasswordAuthenticationToken) authentication);
} catch (AuthenticationException exception) {
    // 进行附加认证校验
    if (cacheWasUsed) {
        // There was a problem, so try again after checking
        // we're using latest data (i.e. not from the cache)
        cacheWasUsed = false;
        user = retrieveUser(username,
            (UsernamePasswordAuthenticationToken) authentication);
        preAuthenticationChecks.check(user);
        additionalAuthenticationChecks(user,
            (UsernamePasswordAuthenticationToken) authentication);
    } else {
        throw exception;
    }
}

// 进行后认证校验
postAuthenticationChecks.check(user);

if (!cacheWasUsed) {
    this.userCache.putUserInCache(user);
}

Object principalToReturn = user;

if (forcePrincipalAsString) {
    principalToReturn = user.getUsername();
}

// 当通过时，创建success的认证，否则在之前抛出相应的异常
return createSuccessAuthentication(principalToReturn, authentication, user);
}
```

认证失败异常处理：

- throw new LockedException(messages.getMessage("AbstractUserDetailsAuthenticationProvider.locked", defaultMessage: "User account is locked"));
- if (user.isEnabled()) { logger.debug("User account is disabled"); throw new DisabledException(messages.getMessage("AbstractUserDetailsAuthenticationProvider.disabled", defaultMessage: "User is disabled"));
- if (user.isAccountExpired()) { logger.debug("User account is expired"); throw new AccountExpiredException(messages.getMessage("AbstractUserDetailsAuthenticationProvider.expired", defaultMessage: "User account has expired"));
- if (authentication.getCredentials() == null) { logger.debug("Authentication failed: no credentials provided"); throw new BadCredentialsException(messages.getMessage("AbstractUserDetailsAuthenticationProvider.badCredentials", defaultMessage: "Bad credentials"));
- String presentedPassword = authentication.getCredentials().toString(); if (!passwordEncoder.matches(presentedPassword, userDetails.getPassword())) { logger.debug("Authentication failed: password does not match stored value"); throw new BadCredentialsException(messages.getMessage("AbstractUserDetailsAuthenticationProvider.badCredentials", defaultMessage: "Bad credentials"));
- private class DefaultPostAuthenticationChecks implements UserDetailsChecker { public void check(UserDetails user) { if (user.isCredentialsNonExpired()) { logger.debug("User account credentials have expired"); throw new CredentialsExpiredException(messages.getMessage("AbstractUserDetailsAuthenticationProvider.credentialsExpired", defaultMessage: "User account credentials have expired"));

在上图中，我们可以看到认证校验分为 前校验、附加校验和后校验，如果任何一个校验出错，就会抛出相应的异常。所有校验都通过后，调用 createSuccessAuthentication() 返回认证信息

```

public AbstractAuthenticationToken(Collection<? extends GrantedAuthority> authorities) {
    if (authorities == null) {
        this.authorities = AuthorityUtils.NO_AUTHORITIES;
        return;
    }
    for (GrantedAuthority a : authorities) {
        if (a == null) {
            throw new IllegalArgumentException(
                "Authorities collection cannot contain any null elements");
        }
    }
    ArrayList<GrantedAuthority> temp = new ArrayList<>(
        authorities.size());
    temp.addAll(authorities);
    this.authorities = Collections.unmodifiableList(temp);
}

protected Authentication createSuccessAuthentication(Object principal,
    Authentication authentication, UserDetails user) {
    // Ensure we return the original credentials the user supplied,
    // so subsequent attempts are successful even with encoded password
    // Also ensure we return the original getDetails(), so that future
    // authentication events after cache expiry contain the details
    UsernamePasswordAuthenticationToken result = new UsernamePasswordAuthenticationToken(
        principal, authentication.getCredentials(),
        authoritiesMapper.mapAuthorities(user.getAuthorities()));
    result.setDetails(authentication.getDetails());
    return result;
}

public UsernamePasswordAuthenticationToken(Object principal, Object credentials,
    Collection<? extends GrantedAuthority> authorities) {
    super(authorities);
    this.principal = principal;
    this.credentials = credentials;
    super.setAuthenticated(true); // must use super, as we override
}

```

初始化 authorities

初始化 authorities

此时认证成功，将 authenticated 设为 true

在 createSuccessAuthentication 方法中，我们发现它重新 new 了一个 UsernamePasswordAuthenticationToken，因为到这里认证已经通过了，所以将 authorities 注入进去，并设置 authenticated 为 true，即需要认证。

4.至此认证信息就被传递回 UsernamePasswordAuthenticationFilter 中，在 UsernamePasswordAuthenticationFilter 的父类 AbstractAuthenticationProcessingFilter 的 doFilter() 中，会根据认证的成功或者失败调用相应的 handler

```

try {
    authResult = attemptAuthentication(request, response, chain, authResult);
    if (authResult == null) {
        // return immediately as subclass has
        // authentication
        return;
    }
    sessionStrategy.onAuthentication(authResult);
} catch (InternalAuthenticationServiceException failed) {
    logger.error(
        "An internal error occurred while trying to authenticate the user.",
        failed);
    unsuccessfulAuthentication(request, response, failed);
    return;
} catch (AuthenticationException failed) {
    // Authentication failed
    unsuccessfulAuthentication(request, response, failed);
    return;
}

// Authentication success
if (continueChainBeforeSuccessfulAuthentication) {
    chain.doFilter(request, response);
}

successfulAuthentication(request, response, chain, authResult);
}

protected void unsuccessfulAuthentication(HttpServletRequest request,
    HttpServletResponse response, AuthenticationException failed)
    throws IOException, ServletException {
    SecurityContextHolder.clearContext();
    if (logger.isDebugEnabled()) {
        logger.debug("Authentication request failed: " + failed.toString(), failed);
        logger.debug("Updated SecurityContextHolder to contain null Authentication");
        logger.debug("Delegating to authentication failure handler " + failureHandler);
    }
    rememberMeServices.loginFail(request, response);
    failureHandler.onAuthenticationFailure(request, response, failed);
}

protected void successfulAuthentication(HttpServletRequest request,
    HttpServletResponse response, FilterChain chain, Authentication authResult)
    throws IOException, ServletException {
    if (logger.isDebugEnabled()) {
        logger.debug("Authentication success. Updating SecurityContextHolder to contain: "
            + authResult);
    }
    SecurityContextHolder.getContext().setAuthentication(authResult);
    rememberMeServices.loginSuccess(request, response, authResult);

    // Fire event
    if (this.eventPublisher != null) {
        eventPublisher.publishEvent(new InteractiveAuthenticationSuccessEvent(
            authResult, this.getClass()));
    }
    successHandler.onAuthenticationSuccess(request, response, authResult);
}

```

调用设置的 failureHandler() 逻辑

调用设置的 successHandler() 逻辑

这里调用的 handler 实际就是我们在配置文件中配置的 successHandler() 和 failureHandler()。

很多小伙伴就怕源码分析过程，但是这类问题是我们绕不过去的一道坎，面试官通过问这种原理性问题可以了解到面试者对于所学框架或技术的理解程度，也在考验我们平时是否跟踪源码。

Spring Security 认证流程回答示范

为了让小伙伴吗，更加容易记忆，给大家准备了文字版认证流程说明。

- (1) 用户在浏览器中随意输入一个URL
- (2) Spring Security 会判断当前是否已经被认证（登录）如果已经认证，正常访问URL。如果没有被认证跳转到（AbstractAuthenticationFilterConfigurer的loginPage属性所对应的值）登录页面。
- (3) 在登录页面中输入用户名和密码后，点击登录按钮后，执行登录
- (4) 如果url是登录的url（AbstractAuthenticationFilterConfigurer的loginProcessingUrl）执行认证流程。否则重新显示认证页面。
- (5) 执行登录流程时通过过滤器UsernamePasswordAuthenticationFilter进行过滤，取出用户名和密码，放入到容器UsernamePasswordAuthenticationToken中。之后交给认证管理器AuthenticationManager去调用认证提供器AuthenticationProvider。
- (6) 由提供器AuthenticationProvider调用自定义登录逻辑UserDetailsService，返回结果UserDetails
- (7) 使用密码解析器PasswordEncoder，对UserDetails中的密码和客户端传递过来的密码进行匹配，如果匹配失败抛出异常BadCredentialsException，如果认证成功，跳转到对应页面