之前简单搭建了个Spring Boot + Spring Security的项目，认证的用户名和密码都是由Spring Security生成。Spring Security支持我们自定义认证的过程，如处理用户信息获取逻辑，使用我们自定义的登录页面替换Spring Security默认的登录页及自定义登录成功或失败后的处理逻辑等。

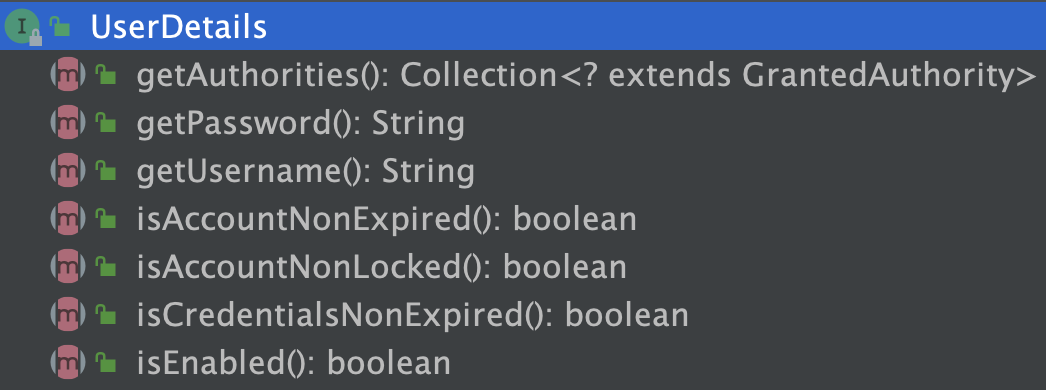
#### 1、自定义认证过程

自定义认证的过程需要实现Spring Security提供的UserDetailsService接口：

public interface UserDetailsService {

UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;  
}

该接口只有一个抽象方法loadUserByUsername：



* getAuthorities获取用户包含的权限，返回权限集合，权限是一个继承了GrantedAuthority的对象；
* getPassword和getUsername用于获取密码和用户名；
* isAccountNonExpired方法返回boolean类型，用于判断账户是否未过期，未过期返回true反之返回false；
* isAccountNonLocked方法用于判断账户是否未锁
* isCredentialsNonExpired用于判断用户凭证是否没过期，即密码是否未过期；
* isEnabled方法用于判断用户是否可用。

实际中可以自定义UserDetails接口的实现类，也可以直接使用Spring Security提供的UserDetails接口实现类org.springframework.security.core.userdetails.User。

下面来实现UserDetailsService接口里的loadUserByUsername方法。

1）首先创建一个User对象，用于存放模拟的用户数据（实际从数据库获取）。

public class User implements Serializable {

private static final long *serialVersionUID* = 3497935890426858541L;  
  
 private String userName;  
  
 private String password;  
  
 private boolean accountNonExpired = true;  
  
 private boolean accountNonLocked = true;  
  
 private boolean credentialsNonExpired = true;  
  
 private boolean enabled = true;

2）创建UserDetailsServiceImpl实现UserDetailsService：

@Configuration

public class UserDetailsServiceImpl implements UserDetailsService {  
  
 @Autowired  
 private PasswordEncoder passwordEncoder;  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
   
 User user = new User();  
 user.setUserName(username);  
 user.setPassword(this.passwordEncoder.encode("123456"));  
  
 return new org.springframework.security.core.userdetails.User(username, user.getPassword(), user.isEnabled(),  
 user.isAccountNonExpired(), user.isCredentialsNonExpired(),  
 user.isAccountNonLocked(), AuthorityUtils.*commaSeparatedStringToAuthorityList*("admin"));  
 }  
}

这里使用org.springframework.security.core.userDetails.User类，包7个参数的构造器，其还包含一个三个参数的构造器User(String username, String password,Collection<? extends GrantedAuthority> authorities)，由于权限参数不能为空，所以先使用AuthorityUtils.commaSeparatedStringToAuthorityList方法模拟一个admin的权限，该方法可以将逗号分隔的字符串转换为权限集合。

还注入了PasswordEncoder对象，该对象用于密码加密，注入前需要手动配置。

3）创建BrowserSecurityConfig类继承WebSecurityConfigurerAdapter。

@Configuration

public class BrowserSecurityConfig extends WebSecurityConfigurerAdapter {  
  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
 ...  
}

PasswordEncoder是一个加密接口，BCryptPasswordEncoder()是Spring提供的一个实现方法，也可以自定义。

4）启动项目，访问http://localhost:8080/login，便可以使用任意用户名以及123456作为密码登录系统。

#### 2、替换默认登录页

直接在src/main/resources/resources目录下定义一个login.html（不需要Controller跳转）：



要怎么做才能让Spring Security跳转到自定义的登录页面呢？只需要在BrowserSecurityConfig的configure中添加配置：

@Override

protected void configure(HttpSecurity http) throws Exception {  
 http.formLogin() // 表单登录  
 .loginPage("/login.html")   
 .loginProcessingUrl("/login")  
 .and()  
 .authorizeRequests() // 授权配置  
 .antMatchers("/login.html").permitAll()  
 .anyRequest() // 所有请求  
 .authenticated(); // 都需要认证  
}

.loginPage()指定了跳转到登录页面的请求URL；.loginProcessingUrl()对应登录页面form表单的action=”/login”；.antMathcers().permitAll()表示跳转登录页面的请求不被拦截，否则会进入无线循环。

这时候启动项目访问<http://localhost:8080/hello>，会看到页面被重定向到<http://localhost:8080/login.html>。输入用户名密码页面报错，先把CSRF攻击防御关了，修改BrowserSecurityConfig的configure。重启便可正常登录。

假如现在有这样一个需求：在未登录的情况下，当用户访问html资源的时候跳转到登录页，否则返回JSON格式数据，状态码为401。将loginPage的URL改为/authentication/require，并且在antMatchers方法中加入该URL，让其免拦截：

@Override

protected void configure(HttpSecurity http) throws Exception {  
 http.formLogin()  
 .loginPage("/authentication/require") // 登录跳转 URL  
 .loginProcessingUrl("/login") // 处理表单登录 URL  
 .and()  
 .authorizeRequests() // 授权配置  
 .antMatchers("/authentication/require", "/login.html").permitAll() // 登录跳转 URL 无需认证  
 .anyRequest() // 所有请求  
 .authenticated() // 都需要认证  
 .and().csrf().disable();  
}

然后定义一个控制器BrowserSecurityController，处理这个请求：

|  |
| --- |
| @RestController  public class BrowserSecurityController {  private RequestCache requestCache = new HttpSessionRequestCache();  private RedirectStrategy redirectStrategy = new DefaultRedirectStrategy();   @GetMapping("/authentication/require")  @ResponseStatus(HttpStatus.UNAUTHORIZED)  public String requireAuthentication(HttpServletRequest request, HttpServletResponse response) throws IOException {  SavedRequest savedRequest = requestCache.getRequest(request, response);  if (savedRequest != null) {  String targetUrl = savedRequest.getRedirectUrl();  if (StringUtils.endsWithIgnoreCase(targetUrl, ".html"))  redirectStrategy.sendRedirect(request, response, "/login.html");  }  return "访问的资源需要身份认证！";  } } |

* HttpSessionRequestCache为Spring Security提供的用于缓存请求的对象，通过调用它的getRequest方法可以获取到本次请求的HTTP信息。
* DefaultRedirectStrategy的sendRedirect为Spring Security提供的用于处理重定向的方法。

这样当我们访问http://localhost:8080/hello的时候页面便会跳转到http://localhost:8080/authentication/require，并且输出”访问的资源需要身份认证！”，当我们访问http://localhost:8080/hello.html的时候，页面将会跳转到登录页面。

#### 3、处理成功和失败

Spring Security有一套默认的处理登录成功和失败的方法：当用户登录成功时，页面会跳转会引发登录的请求，比如在未登录的情况下访问<http://localhost:8080/hello>，页面会跳转到登录页，登录成功后再跳转回来；登录失败时则是跳转到Spring Security默认的错误提示页面。下面通过一些自定义配置来替换这套默认的处理机制。

**自定义登录成功逻辑**

实现AuthenticationSuccessHandler接口的onAuthenticationSuccess方法即可：

@Component

public class MyAuthenticationSucessHandler implements AuthenticationSuccessHandler {  
 @Override  
 public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response,  
 Authentication authentication) throws IOException, ServletException {  
 response.setContentType("application/json;charset=utf-8");  
 response.getWriter().write(mapper.writeValueAsString(authentication));  
 }  
}

Authentication参数既包含了认证请求的一些信息，比如IP，请求的SessionId等，也包含了用户信息，即前面提到的User对象。要使这个配置生效，还要在BrowserSecurityConfig的configure中配置：

.successHandler(authenticationSucessHandler) // 处理登录成功

我们将MyAuthenticationSuccessHandler注入进来，并通过successHandler方法进行配置。登录成功后页面将会输出如下JSON信息：



**自定义登录失败逻辑**

实现AuthenticationFailureHandler的onAuthenticationFailure方法：

@Component

public class MyAuthenticationFailureHandler implements AuthenticationFailureHandler {  
 @Override  
 public void onAuthenticationFailure(HttpServletRequest request, HttpServletResponse response,  
 AuthenticationException exception) throws IOException {  
 }  
}

AuthenticationException参数是一个抽象类，Spring Security根据登录失败的原因封装了许多对应的实现类。不同的失败原因对应不同的异常，比如用户名

