链滴

Springboot 之 Security 前后端分离登录

作者：hjljy
原文链接：https://ld246.com/article/1600073776051
来源网站：链滴

什么是Spring Security

Spring Security是一个功能强大且高度可定制的身份验证和访问控制框架。它是用于保护基于Spring应用程序的实际标准。

Spring Security是一个框架，致力于为Java应用程序提供身份验证和授权。与所有Spring项目一样，pring Security的真正强大之处在于可以轻松扩展以满足自定义要求

官方网站：https://spring.io/projects/spring-security#learn

初识Security

因为之前已经接触过其他的权限框架：shiro 所以很好入门

比较好的入门博文：

Springboot + Spring Security 实现前后端分离登录认证及权限控制

Springboot集成SpringSecurity 附代码

关于SpringBoot应用中集成Spring Security你必须了解的那些事

Spring Boot Security

具体代码实现

通常都是通过自定义UserDetailsService, AuthenticationProvider, AuthenticationManager,UsernamePasswordAuthenticationFilter其中的一种来实现的。

最常见的是通过UserDetailsService方式来实现，方便快捷

1 引入Security包

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
```

2 创建Security配置类

```
/**
 * @author 海加尔金鹰 www.hjljy.cn
 * @apiNote websecurtiy权限校验处理
 * @since 2020/9/11
 **/
@Configuration
@EnableWebSecurity
@EnableGlobalAuthentication
public class WebSecurityConfiguration extends WebSecurityConfigurerAdapter {

    /**
     * 描述:
     * http方式走 Spring Security 过滤器链，在过滤器链中，给请求放行，而web方式是不走 Spring
ecurity 过滤器链。
     * 通常http方式用于请求的放行和限制，web方式用于放行静态资源
     **/
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                //用于配置直接放行的请求
                .antMatchers("/login").permitAll()
                //其余请求都需要验证
                .anyRequest().authenticated()
                //授权码模式需要 会弹出默认自带的登录框
                .and().httpBasic()
                //禁用跨站伪造
                .and().csrf().disable();
                //如果项目没有前后端分离，还可以通过 formlogin配置登录相关的页面和请求处理
        // 使用自定义的认证过滤器
        // http.addFilterBefore(new  MyLoginFilter(authenticationManager()),UsernamePasswor
AuthenticationFilter.class);
    }

    /**
     * 描述: 静态资源放行，这里的放行，是不走 Spring Security 过滤器链
     **/
    @Override
    public void configure(WebSecurity web) {
        // 可以直接访问的静态数据
        web.ignoring()
                .antMatchers("/css/**")
                .antMatchers("/404.html")
                .antMatchers("/500.html")
                .antMatchers("/html/**")
                .antMatchers("/js/**");
    }
```

```
    /**
     * 描述：设置授权处理相关的具体类以及加密方式
     */
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
        // 设置不隐藏 未找到用户异常
        provider.setHideUserNotFoundExceptions(true);
        // 用户认证service - 查询数据库的逻辑
        provider.setUserDetailsService(userDetailsService());
        // 设置密码加密算法
        provider.setPasswordEncoder(passwordEncoder());
        auth.authenticationProvider(provider);
    }

    /**
     * 描述: 通过自定义的UserDetailsService 来实现查询数据库用户数据
     **/
    @Override
    @Bean
    protected UserDetailsService userDetailsService() {
        return new UserDetailsServiceImpl();
    }

    /**
     * 描述: 密码加密算法 BCrypt 推荐使用
     **/
    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    /**
     * 描述: 注入AuthenticationManager管理器
     **/
    @Override
    @Bean
    public AuthenticationManager authenticationManager() throws Exception {
        return super.authenticationManager();
    }
}
```

3 创建UserInfo类

```
/**
 * @author 海加尔金鹰
 * @apiNote 用户信息类 主要是提供给验证框架使用，并将用户基本信息保存到这个类
 * @since 2020/9/11
 **/
public class UserInfo extends User {

    private static final long serialVersionUID = 1L;
```

```
    /**
     * 描述: 可以添加自定义的用户属性
     * 用户邮箱
     **/
    private String email;
    /**
     * 描述: 用户ID
     **/
    private String userId;

    public UserInfo(String username, String password, Collection<? extends GrantedAuthority>
authorities) {
        super(username, password, authorities);
    }
    public UserInfo(String username, String password, String userId,Collection<? extends Grant
dAuthority> authorities) {
        super(username, password, authorities);
        this.userId=userId;
    }

    public UserInfo(String username, String password, boolean enabled, boolean accountNonE
pired, boolean credentialsNonExpired, boolean accountNonLocked, Collection<? extends Gra
tedAuthority> authorities) {
        super(username, password, enabled, accountNonExpired, credentialsNonExpired, accoun
NonLocked, authorities);
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getUserId() {
        return userId;
    }

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

4 实现UserDetailsService

```
/**
 * @author 海加尔金鹰
 * @apiNote 用户具体验证类
 * @since 2020/9/11
 **/
public class UserDetailsServiceImpl implements UserDetailsService {
    /**
```

```
     * 这里根据传进来的用户账号进行用户信息的构建
     * 通常的做法是
     *  1 根据username查询数据库对应的用户信息
     *  2 根据用户信息查询出用户权限信息  例如菜单添加权限  sys:menu:add
     *  3 根据用户账号，密码，权限构建对应的UserDetails对象返回
     * 这里实际上是没有进行用户认证功能的，真正的验证是在UsernamePasswordAuthenticationFil
er对象当中
     * UsernamePasswordAuthenticationFilter对象会自动根据前端传入的账号信息和UserDetails
象对比进行账号的验证
     * 通常情况下，已经满足常见的使用常见，不过如果有特殊需求，需要使用自己实现的具体认证方
，可以继承UsernamePasswordAuthenticationFilter对象
     * 重写attemptAuthentication 方法和successfulAuthentication方法
     * 最后在WebSecurityConfiguration里面添加自己的过滤器即可
     * @param username 用户账号
     * @return UserInfo
     * @throws UsernameNotFoundException
     */
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundExce
tion {
        //TODO 当前使用测试数据进行测试 需要修改成实际的业务逻辑处理
        //  不限制用户账号。只要密码是123456就可以通过验证 并添加权限
        String password = SecurityUtils.encryptPassword("123456");
        SimpleGrantedAuthority authority = new SimpleGrantedAuthority("sys:menu:add");
        List<GrantedAuthority> authorities =new ArrayList<>();
        authorities.add(authority);
        UserInfo userInfo =new UserInfo(username,password,authorities);
        userInfo.setEmail("hjljy@outlook.com");
        userInfo.setUserId("11111111");
        return userInfo;
    }
}
```

5 提供前端一个登录接口

由于项目进行前后端分离以及Security自带的登录界面不美观，所以需要提供一个登录接口给前端调，该接口需要在配置当中放行，未授权访问需要授权的请求时，会返回401或者403状态码，前端可根据这个进行路由提示处理

```
@RestController
public class LoginController {


    @Autowired
    AuthenticationManager authenticationManager;

    @PostMapping(value = "login")
    public ResultInfo login(@RequestBody Map<String,String> params)  {
        UserInfo userInfo = SecurityUtils.login(params.get("username"), params.get("password"),
authenticationManager);
        return ResultInfo.success(userInfo);
    }
}
```

6 SecurityUtils 工具类

```
@Slf4j
public class SecurityUtils {

    /**
     * 描述根据账号密码进行调用security进行认证授权 主动调
     * 用AuthenticationManager的authenticate方法实现
     * 授权成功后将用户信息存入SecurityContext当中
     * @param username 用户名
     * @param password 密码
     * @param authenticationManager 认证授权管理器,
     * @see  AuthenticationManager
     * @return UserInfo  用户信息
     */
    public static UserInfo login(String username, String password, AuthenticationManager auth
ntticationManager) throws AuthenticationException {
        //使用security框架自带的验证token生成器  也可以自定义。
        UsernamePasswordAuthenticationToken token =new UsernamePasswordAuthentication
oken(username,password );
        Authentication authenticate = authenticationManager.authenticate(token);
        SecurityContextHolder.getContext().setAuthentication(authenticate);
        UserInfo userInfo = (UserInfo) authenticate.getPrincipal();
        return userInfo;
    }

    /**
     * 获取当前登录的所有认证信息
     * @return
     */
    public static Authentication getAuthentication(){
        SecurityContext context = SecurityContextHolder.getContext();
        return context.getAuthentication();
    }

    /**
     * 获取当前登录用户信息
     * @return
     */
    public static UserInfo getUserInfo(){
        Authentication authentication = getAuthentication();
        if(authentication!=null){
            Object principal = authentication.getPrincipal();
            if(principal!=null){
                UserInfo userInfo = (UserInfo) authentication.getPrincipal();
                return userInfo;
            }
        }
        throw new BusinessException();
    }

    /**
     * 获取当前登录用户ID
     * @return
```

```
     */
    public static String getUserId(){
        UserInfo userInfo = getUserInfo();
        return userInfo.getUserId();
    }

    /**
     * 生成BCryptPasswordEncoder密码
     *
     * @param password 密码
     * @return 加密字符串
     */
    public static String encryptPassword(String password) {
        BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
        return passwordEncoder.encode(password);
    }
}
```

7 登录验证

上述流程完毕之后，可以通过工具发送HTTP测试一下登录接口

```
POST http://localhost:8090/login
Content-Type: application/json

{
  "username": "测试账号",
  "password": 123456
}
```

成功返回账号信息

```
{
  "code": 0,
  "msg": "操作成功",
  "data": {
    "password": null,
    "username": "测试账号",
    "authorities": [
      {
        "authority": "sys:menu:add"
      }
    ],
    "accountNonExpired": true,
    "accountNonLocked": true,
    "credentialsNonExpired": true,
    "enabled": true,
    "email": "hjljy@outlook.com",
    "userId": "11111111"
  }
}
```

总结

总的来说入门还是很简单的，网上的资料也比较多，但是大多数的前后端分离都是自定义登录界面，是接口分离。还有就是基本上security 会搭配oauth2使用进行权限验证。

原文链接：Springboot 之 Security 前后端分离登录