认证和授权：前后端分离状态下使用Spring Security实现安全访问控制

Java精选 2022-10-09 08:00 发表于北京

收录于合集
#SpringSecurity
2个

本文使用的springboot版本是2.1.3.RELEASE

一、简要描述

默认情况下，spring security登录成功或失败，都会返回一个302跳转。登录成功跳转到主页，失败跳转到登录页。如果未认证直接访问受保护资源也会跳转到登录页 。

而在前后端分离项目中，前后端是通过json数据进行交互，前端通过ajax请求和后端进行交互，ajax是无法处理302跳转的，所以我们希望不管是未登录还是登录成功，spring security都给前端返回json数据,而前端自己根据返回结果进行逻辑控制。

springsecurity默认采用的是表单登录,而我们希望的登录流程是这样的：

(1) 前端带着用户名和密码用ajax请求登录，认证成功后返回一个token值给前端

(2) 下次请求时在请求头中携带这个token,后端校验这个token通过后放行请求,否则提示未登录(返回json数据)

二、配置让springsecurity返回 json数据

2.1 未登录时访问受限资源的处理

未登录时访问资源，请求会被FilterSecurityInterceptor这个过滤器拦截到，然后抛出异常,这个异常会被

ExceptionTranslationFilter这个过滤器捕获到，并最终交给AuthenticationEntryPoint接口的commence方法处理。

所以处理办法是自定义一个AuthenticationEntryPoint的实现类并配置到springsecurity中

```
/**
 * 未登录时访问受限资源的处理方式
 */
```

```
public class UnLoginHandler implements AuthenticationEntryPoint {
    @Override
    public void commence(HttpServletRequest request, HttpServletResponse response, Authenticat
        ObjectMapper mapper = new ObjectMapper();
        ObjectNode objectNode = mapper.createObjectNode();
        if(authException instanceof BadCredentialsException){
            //账号或密码错误
            objectNode.put("code", "501");
            objectNode.put("message", "账号或者密码错误");
        }else {
            objectNode.put("code", "500");
            objectNode.put("message", "未登录或token无效");
        }

        response.setHeader("Content-Type", "application/json;charset=UTF-8");
        response.getWriter().print(objectNode);
    }
}
```

配置到spring security中，

```
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    ...省略其他配置，公众号Java精选，有惊喜
        //安全配置
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        ...省略其他配置
        //设置未登录或登录失败时访问资源的处理方式
        http.exceptionHandling().authenticationEntryPoint(new UnLoginHandler());
        ...
    }
}
```

2.2 访问资源权限不足时的处理

当一个已登录用户访问了一个没有权限的资源时，springsecurity默认会重定向到一个403页面。可以通过自己实现 AccessDeniedHandler接口然后配置到springsecurity中来自定义

```
/**
 * 当前登录的用户没有权限访问资源时的处理器
 */
public class NoAccessDeniedHandler implements AccessDeniedHandler {
```

```
    @Override
    public void handle(HttpServletRequest request, HttpServletResponse response, AccessDeniedE
        ObjectMapper mapper = new ObjectMapper();
        ObjectNode objectNode = mapper.createObjectNode();
        objectNode.put("code","500");
        objectNode.put("message","访问失败，权限不够");
        response.setHeader("Content-Type","application/json;charset=UTF-8");
        response.getWriter().print(objectNode);
    }
}
```

配置到springsecurity中

```
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    ...省略其他配置
        //安全配置
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        ...省略其他配置
        //设置权限不足,无法访问当前资源时的处理方式
        http.exceptionHandling().accessDeniedHandler(new NoAccessDeniedHandler());
        ...
    }
}
```

这样配置后，未登录，登录失败，权限不足这些场景下springsecurity就会返回json数据给前端。springsecurity系列技术文章，待更新中：https://www.yoodb.com/spring/spring-annotate.html

三、如何发token

这一节来解决发token的问题。现在已经去掉了表单登录的功能，那如何让springsecurity验证账号和密码并创建token呢。

可以自定义一个接口给前端请求，用来发token,前端提交账号和密码到这个接口，在其中调用springsecurity的认证管理器来认证账号密码，认证成功后创建一个token返回给前端

```
@RestController
@RequestMapping("/authenticate")
public class AuthenticationController {

    private final static ObjectMapper MAPPER=new ObjectMapper();
```

```
    //注入springsecurity的认证管理器
    @Autowired
    private AuthenticationManager authenticationManager;

    /**
     * 创建token
     * @return
     */
    @PostMapping("/applyToken")
    public JsonNode applyToken(@RequestBody UserDto userDto){
        ObjectNode tokenNode = MAPPER.createObjectNode();
        //1.创建UsernamePasswordAuthenticationToken对象
        UsernamePasswordAuthenticationToken authenticationToken=new UsernamePasswordAuthentica
        //2.交给认证管理器进行认证
        Authentication authenticate = authenticationManager.authenticate(authenticationToken);

        if(null!=authenticate){
            //认证成功,生成token返回给前端
            String token = JwtUtils.createToken(userDto.getUsername());
            if(StringUtils.isEmpty(token)){
                tokenNode.put("code","401");
                tokenNode.put("message","生成token失败");
            }else {
                tokenNode.put("code","200");
                tokenNode.put("token", token);
                tokenNode.put("message","success");
            }
            tokenNode.put("code","200");
            tokenNode.put("token", JwtUtils.createToken(userDto.getUsername()));
            tokenNode.put("message","success");
            return tokenNode;
        }else{
            tokenNode.put("code","401");
            tokenNode.put("message","登录失败");
        }
        return tokenNode;
    }
}
```

其中JwtUtils是一个自定义的jwt 工具类,提供了生成token和验证token的功能

四、如何让springsecurity验证token

上边实现了发token的功能，那如何让springsecurity验证这个token,并放行请求。可以自定义一个过滤器，在springsecurity的登录过滤器之前先拦截请求，然后进行token，如果验证通过了就把当前用户设置到SecurityContextHolder中，这样就完成了验证和登录。

自定义过滤器

```
/**
 * 验证请求携带的token是否有效
 */
@Component
public class TokenVerifyFilter extends GenericFilterBean {

    @Autowired
    private UserDetailsService userDetailsService;

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, Filte

        try {
            HttpServletRequest request = (HttpServletRequest) servletRequest;
            //从请求头中获取token
            String token = request.getHeader("Authorization-Token");
            if (StringUtils.hasText(token)) {
                //从token中解析用户名
                String username = JwtUtils.getUserInfo(token);
                //查询当前用户
                if(!StringUtils.isEmpty(username)){
                    UserDetails userDetails = userDetailsService.loadUserByUsername(username);
                    if(null!=userDetails){
                        //查询不到表示用户不存在
                        //从token中获取用户信息封装成 UsernamePasswordAuthenticationToken
                        UsernamePasswordAuthenticationToken authenticationToken = new Username
                        //设置用户信息
 SecurityContextHolder.getContext().setAuthentication(authenticationToken);
                    }
                }
            }
        } catch (Exception e) {
            //登录发生异常,但要继续走其余过滤器的逻辑
            e.printStackTrace();
        }
        //继续执行springsecurity的过滤器
        filterChain.doFilter(servletRequest, servletResponse);
    }
}
```

把这个过滤器设置到UsernamePasswordAuthenticationFilter之前。面试宝典：https://www.yoodb.com

完整的springsecurity安全配置如下：

```
/**
 * 配置springsecurity
 */
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {


    //用户配置,
    @Bean
    public UserDetailsService userDetailsService(){
        //在内存中配置用户
        InMemoryUserDetailsManager manager=new InMemoryUserDetailsManager();
        manager.createUser(User.withUsername("lyy").password("123").authorities("ROLE_P1").bui
        manager.createUser(User.withUsername("zs").password("456").authorities("ROLE_P2").buil
        return manager;
    }

    //配置自定义的对token进行验证的过滤器
    @Autowired
    private TokenVerifyFilter tokenVerifyFilter;

    //密码加密方式配置
    @Bean
    public PasswordEncoder passwordEncoder(){
        return NoOpPasswordEncoder.getInstance();
    }

    //安全配置
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable();
        //匹配路径时越具体的路径要先匹配
        http.authorizeRequests().antMatchers("/","/index.html").permitAll();
        //放行申请token的url
        http.authorizeRequests().antMatchers("/authenticate/**").permitAll();
        //需要p1权限才能访问
        http.authorizeRequests().antMatchers("/resource/r1").hasRole("P1");
        //需要p2权限才能访问
        http.authorizeRequests().antMatchers("/resource/r2").hasRole("P2")
        .antMatchers("/resource/r3").hasRole("P3");//需要p3权限才能访问
        http.authorizeRequests().anyRequest().authenticated();
```

```
        http.formLogin().disable();//禁用表单登录
        //设置未登录或登录失败时访问资源的处理方式
        http.exceptionHandling().authenticationEntryPoint(new UnLoginHandler());
        //设置权限不足,无法访问当前资源时的处理方式
        http.exceptionHandling().accessDeniedHandler(new NoAccessDeniedHandler());
        http.addFilterBefore(tokenVerifyFilter, UsernamePasswordAuthenticationFilter.class);
        //设置不使用session,无状态
        http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    }

    /**
     * 配置认证管理器:
     * @return
     * @throws Exception
     */
    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
}
```

五、总结

按上边这样配置后，前端向先请求发token的接口获取一个token，然后在每次访问后端时都在请求头中带上这个token,后端验证了这个token后就会放行请求。

完整的示例工程源码：

https://gitee.com/zhituaishangc/cnblog-springsecurity-study/tree/master/cnblog-springsecurity05-test

> 作者：程序晓猿
> https://www.cnblogs.com/chengxuxiaoyuan/p/14020326.html

------ 特别推荐 ------

特别推荐：专注分享最前沿的技术与资讯，为弯道超车做好准备及各种开源项目与高效率软件的公众号，**「大咖笔记」**，专注挖掘好东西，非常值得大家关注。**点击下方公众号卡片关注**。

大咖笔记

关注最前沿的技术与资讯，为弯道超车做好准备！

2篇原创内容

公众号

点击“阅读原文”，了解更多精彩内容！文章有帮助的话，点在看，转发吧！

收录于合集 #SpringSecurity 2

下一篇 · Spring Security 学习笔记，看了必懂！

阅读原文

喜欢此内容的人还喜欢

node应用故障定位顶级技巧—动态追踪技术[Dynamic Trace]

元语言

40 个 SpringBoot 常用注解：让生产力爆表！

猿大侠

Spring Boot 实现万能文件在线预览，已开源，真香！！

开源前线