超全的springboot+springsecurity实现前后端分离简单实现！

java1234 今天

1、前言部分

1.1、唠嗑部分(如何学习?)

看springsecurtiy原理图的时候以为洒洒水，妈的，结果自己动手做的时候一窍不通，所以一定不要眼高手低，实践出真知！通过各种方式学习springsecurity，在**B站、腾讯课堂、网易课堂、慕课网**没有springsecurity的前后端分离的教学视频，那我就去**csdn**去寻找springsecurity博客，发现几个问题：

- 要么就是前后端不分离，要么就是通过内存方式读取数据，而不是通过数据库的方式读取数据，要么就是大佬们给的代码不全、把代码讲的太绕，关键部分没有注释。
- 讲的例子不那么通俗易懂，不利于新手的学习。
- 代码本身有bug，或者就没有我想要实现的效果。

实在不行我又跑去**github**上找开源项目学习，**github**由于是外国网站，国内访问速度有点慢！！那就用国内的**gitee**吧，**gitee**上的开源项目都是结合实战项目的，代码逻辑也比较复杂，我对项目的业务逻辑没什么了解，感觉不适合我。我这一次选择比较反人性的方式去学习，就是**手撕源码和看官方文档**。老实讲，刚开始看**源码**和**官方文档**特别难受，并且看不进去，那些springsecurity的类还有接口名字又臭又长，这时我就下载源码，**源码的注释多的就像一本书，非常详细且权威**。

当然别指望看几遍就能看懂，我看这些注释、源码、博客看了10几遍甚至20几遍才看懂，每次去看都有不同的收获！！！

此文章截图水平不高、理解为主、欣赏为辅！！内容有点多，每一步都有详细解析，请耐心看完，看不懂可以多看几遍。。

1.2、技术支持

jdk 1.8、springboot 2.3.4、mybatis-plus 3.4.1、mysql 5.5、springsecurity 5.3.4、springmvc、lombok简化entity代码，不用你去写get、set方法，全部自动生成、gson 2.8.2 将json对象转化成json字符串

1.3、预期实现效果图

未登录时访问指定资源， 返回未登录的json字符串， index是我在controller层写的一个简单接口，返回index字符串

输入账号错误，返回用户名错误的json字符串，需说明一点，/login是springsecurity封装好的接口，无须你在controller写login接口，/logout也同理。

输入密码错误，返回密码错误的json字符串

登录成功， 返回登录成功的json字符串并返回cookie

登录成功并且拥有权限访问指定资源， 返回资源相关数据的json字符串

登录成功但无权限访问指定资源时，返回权限不足的json字符串

异地登录，返回异地登录，强制下线的json字符串， **测试的基础是要在两台不同的机器上登录，然后访问/index。**

注销成功，返回注销成功的json字符串并删除cookie

2、核心部分

2.1、springsecurity原理解释:

springsecurity最重要的两个部分: **authentication(认证) 和 authorization(授权)**

认证: 就是判定你是什么身份，管理员还是普通人

授权: 什么样的身份拥有什么样的权利。

简单理解: 自定义配置登录成功、登陆失败、注销成功目标结果类，并将其注入到springsecurity的配置文件中。如何认证、授权交给AuthenticationManager去作

复杂理解:

(1)用户发起表单登录请求后，首先进入 `UsernamePasswordAuthenticationFilter`，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 `SelfAuthenticationProvider`，该类是 **AuthenticationProvider** 的实现类：你可以在该类的 `Authentication authenticate(Authentication authentication)` 自定义认证逻辑，然后在该类中通过调用`UserDetails loadUserByUsername(account)` 去获取数据库用户信息并验证，然后创建 `UsernamePasswordAuthenticationToken` 并将权限、用户个人信息注入到其中，并通过`setAuthenticated(true)` 设置为需要验证。

(3) 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**：所谓的**handler**就是我们注入到**springsecurity**配置文件的**handler。**

2.2、踩坑集锦

- **访问/login时必须要用post方法！，访问的参数名必须为username和password**
- **访问/logout时即可用post也可用get方法！**
- **//springsecurity配置文件中的hasRole("")不能以ROLE开头，比如ROLE_USER就是错的，springsecurity会默认帮我们加上，但数据库的权限字段必须是ROLE_开头，否则读取不到**

```
.antMatchers("/index").hasRole("USER")
.antMatchers("/hello").hasRole("ADMIN")
```

2.3、代码部分

pom依赖文件

```
<dependencies>
        <!--转换成json字符串的工具-->
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.8.2</version>
        </dependency>
        <!--springboot集成web操作7-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!--springsecurity-->
        <dependency>
```

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <!--mysql驱动-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <!--lombok依赖-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <!--mybatis-plus依赖-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>3.4.1</version>
        </dependency>

        <!--springboot-自带测试工具-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
            <version>5.3.4.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
            <version>5.3.4.RELEASE</version>
        </dependency>
    </dependencies>
```

Msg.java 自定义返回结果集，这个看个人的，怎么开心怎么来！

```
@Data
```

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Msg {
    int code;   //错误码
    String Message; //消息提示
    Map<String,Object> data=new HashMap<String,Object>();   //数据

    //无权访问
    public static Msg denyAccess(String message){
        Msg result=new Msg();
        result.setCode(300);
        result.setMessage(message);
        return result;
    }

    //操作成功
    public static Msg success(String message){
        Msg result=new Msg();
        result.setCode(200);
        result.setMessage(message);
        return result;
    }

    //客户端操作失败
    public static Msg fail(String message){
        Msg result=new Msg();
        result.setCode(400);
        result.setMessage(message);
        return result;
    }

    public Msg add(String key,Object value){
        this.data.put(key,value);
        return this;
    }
}
```

User.java ， 此类是entity实体类

```
@Data
public class User implements Serializable {

    private Integer id;

    private String account;

    private String password;

    private String role;
```

```
}
```

UserMapper.java ，此接口继承 BaseMapper<T>类，而**BaseMapper<T>类** 封装了大量的sql，极大程度简化了程序员sql语句的书写.

```
@Repository
public interface UserMapper extends BaseMapper<User> {

}
```

正常情况下要写UserService.java接口，但是此文章只是用于演示效果，就没书写了

```
public interface UserService{

}
```

UserServiceImpl.java，使其实现 UserDetailsService接口，从而去获取数据库用户信息，**详细解析请看注释部分。**

```
@Service
public class UserServiceImpl extends ServiceImpl<UserMapper, User> implements UserService,UserDeta

    @Autowired
    UserMapper userMapper;

    //加载用户
    @Override
    public UserDetails loadUserByUsername(String s) throws UsernameNotFoundException {
        //mybatis-plus帮我们写好了sql语句，相当于 select * from user where account ='${account}'
        QueryWrapper<User> wrapper=new QueryWrapper<>();
        wrapper.eq("account",s);
        User user=userMapper.selectOne(wrapper);    //user即为查询结果
        if(user==null){
            throw new UsernameNotFoundException("用户名错误！！");
        }

        //获取用户权限，并把其添加到GrantedAuthority中
        List<GrantedAuthority> grantedAuthorities=new ArrayList<>();
        GrantedAuthority grantedAuthority=new SimpleGrantedAuthority(user.getRole());
```

```
        grantedAuthorities.add(grantedAuthority);

        //方法的返回值要求返回UserDetails这个数据类型，  UserDetails是接口，找它的实现类就好了
        //new org.springframework.security.core.userdetails.User(String username,String password,(
        return new org.springframework.security.core.userdetails.User(s,user.getPassword(),grantec
    }
}
```

UserController.java 就是普通的controller

```
@RestController
public class UserController {
    @GetMapping("index")
    public String index(){
        return "index";
    }

    @GetMapping("hello")
    public String hello(){
        return "hello";
    }
}
```

AuthenticationEnryPoint .java 自定义未登录的处理逻辑

```
@Component
public class AuthenticationEnryPoint implements AuthenticationEntryPoint {

    @Autowired
    Gson gson;

    //未登录时返回给前端数据
    @Override
    public void commence(HttpServletRequest request, HttpServletResponse response, Authentication
        Msg result=Msg.fail("需要登录!!");
        response.setContentType("application/json;charset=utf-8");
        response.getWriter().write(gson.toJson(result));
    }
}
```

AuthenticationFailure.java 自定义登录失败时的处理逻辑

```
//登录失败返回给前端消息
@Component
public class AuthenticationFailure implements AuthenticationFailureHandler{
    @Autowired
    Gson gson;

    @Override
    public void onAuthenticationFailure(HttpServletRequest request, HttpServletResponse response,
        Msg msg=null;
        if(e instanceof UsernameNotFoundException){
            msg=Msg.fail(e.getMessage());
        }else if(e instanceof BadCredentialsException){
            msg=Msg.fail("密码错误!!");
        }else {
            msg=Msg.fail(e.getMessage());
        }
        //处理编码方式，防止中文乱码的情况
        response.setContentType("text/json;charset=utf-8");
        //返回给前台
        response.getWriter().write(gson.toJson(msg));
    }
}
```

AuthenticationSuccess.java 自定义登录成功时的处理逻辑

```
@Component
public class AuthenticationSuccess implements AuthenticationSuccessHandler{
    @Autowired
    Gson gson;

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response,
        //登录成功时返回给前端的数据
        Msg result=Msg.success("登录成功！！！！！");
        response.setContentType("application/json;charset=utf-8");
        response.getWriter().write(gson.toJson(result));
    }
}
```

AuthenticationLogout.java 自定义注销时的处理逻辑

```
@Component
public class AuthenticationLogout implements LogoutSuccessHandler{
    @Autowired
    Gson gson;

    @Override
    public void onLogoutSuccess(HttpServletRequest request, HttpServletResponse response, Authent
        Msg result=Msg.success("注销成功");
        response.setContentType("application/json;charset=utf-8");
        response.getWriter().write(gson.toJson(result));
    }
}
```

AccessDeny.java 自定义无权限访问时的逻辑处理

```
//无权访问
@Component
public class AccessDeny implements AccessDeniedHandler{
    @Autowired
    Gson gson;

    @Override
    public void handle(HttpServletRequest request, HttpServletResponse response, AccessDeniedExce
        Msg result= Msg.denyAccess("无权访问，need Authorities!!");
        response.setContentType("application/json;charset=utf-8");
        response.getWriter().write(gson.toJson(result));
    }
}
```

SessionInformationExpiredStrategy.java 自定义异地登录、账号下线时的逻辑处理

```
@Component
public class SessionInformationExpiredStrategy implements org.springframework.security.web.sessio
    @Autowired
    Gson gson;

    @Override
    public void onExpiredSessionDetected(SessionInformationExpiredEvent event) throws IOException
        Msg result= Msg.fail("您的账号在异地登录，建议修改密码");
        HttpServletResponse response=event.getResponse();
        response.setContentType("application/json;charset=utf-8");
```

```
        response.getWriter().write(gson.toJson(result));
    }
}
```

SelfAuthenticationProvider.java 自定义认证逻辑处理

```
@Component
public class SelfAuthenticationProvider implements AuthenticationProvider{
    @Autowired
    UserServiceImpl userServiceImpl;

    @Autowired
    BCryptPasswordEncoder bCryptPasswordEncoder;

    @Override
    public Authentication authenticate(Authentication authentication) throws AuthenticationExcept
        String account= authentication.getName();     //获取用户名
        String password= (String) authentication.getCredentials();  //获取密码
        UserDetails userDetails= userServiceImpl.loadUserByUsername(account);
        boolean checkPassword= bCryptPasswordEncoder.matches(password,userDetails.getPassword());
        if(!checkPassword){
            throw new BadCredentialsException("密码不正确，请重新登录!");
        }
        return new UsernamePasswordAuthenticationToken(account,password,userDetails.getAuthoritie
    }

    @Override
    public boolean supports(Class<?> aClass) {
        return true;
    }
}
```

SpringsecurityConfig.java是springsecurity的配置，详细解析请看注释！！

```
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true) //开启权限注解,默认是关闭的
public class SpringsecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    AuthenticationEnryPoint authenticationEnryPoint;    //未登录
    @Autowired
    AuthenticationSuccess authenticationSuccess;    //登录成功
    @Autowired
```

```
AuthenticationFailure authenticationFailure;    //登录失败
@Autowired
AuthenticationLogout authenticationLogout;      //注销
@Autowired
AccessDeny accessDeny;      //无权访问
@Autowired
SessionInformationExpiredStrategy sessionInformationExpiredStrategy;    //检测异地登录
@Autowired
SelfAuthenticationProvider selfAuthenticationProvider;      //自定义认证逻辑处理

@Bean
public UserDetailsService userDetailsService() {
    return new UserServiceImpl();
}

//加密方式
@Bean
public BCryptPasswordEncoder bCryptPasswordEncoder() {
    return new BCryptPasswordEncoder();
}

//认证
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.authenticationProvider(selfAuthenticationProvider);
}

//授权
@Override
protected void configure(HttpSecurity http) throws Exception {
    //cors()解决跨域问题，csrf()会与restful风格冲突，默认springsecurity是开启的，所以要disable()关
    http.cors().and().csrf().disable();

    //     /index需要权限为ROLE_USER才能访问    /hello需要权限为ROLE_ADMIN才能访问
    http.authorizeRequests()
            .antMatchers("/index").hasRole("USER")
            .antMatchers("/hello").hasRole("ADMIN")


            .and()
            .formLogin()  //开启登录
            .permitAll()  //允许所有人访问
            .successHandler(authenticationSuccess) // 登录成功逻辑处理
            .failureHandler(authenticationFailure) // 登录失败逻辑处理

            .and()
            .logout()   //开启注销
            .permitAll()    //允许所有人访问
            .logoutSuccessHandler(authenticationLogout) //注销逻辑处理
            .deleteCookies("JSESSIONID")    //删除cookie
```

```
                .and().exceptionHandling()
                .accessDeniedHandler(accessDeny)    //权限不足的时候的逻辑处理
                .authenticationEntryPoint(authenticationEnryPoint)  //未登录是的逻辑处理

                .and()
                .sessionManagement()
                .maximumSessions(1)     //最多只能一个用户登录一个账号
                .expiredSessionStrategy(sessionInformationExpiredStrategy)  //异地登录的逻辑处理
        ;
    }
}
```

application.yml配置文件

```
server:
  port: 80

spring:
  datasource:
    url: jdbc:mysql://localhost:3306/springsecurity_test?characterEncoding=utf8&serverTimezone=UT
    username: root
    password: 123456
    driver-class-name: com.mysql.cj.jdbc.Driver
```

最终结果在上面就有，源码地址：https://gitee.com/liu-wenxin/springsecurity_demo.git ，使用 git clone https://gitee.com/liu-wenxin/springsecurity_demo.git 下载到本地。

粉丝福利：Java从入门到入土学习路线图

长按上方微信二维码 2 秒

感谢点赞支持下哈

阅读原文

喜欢此内容的人还喜欢

PHP8+GO彻底搞定高并发、微服务架构

脚本之家

请不要在Java项目中乱打印日志了，这才是正确姿势，非常实用！

Java基基

一个基于 SpringBoot 开源的小说和漫画在线阅读网站，简洁大方、强烈推荐

Java笔记虾