15 SpringSecurity 集成 thymeleaf

此项目是在 springsecurity-12-database-authorization-method 的基础上进行

15.1 复制并修改工程

复制 springsecurity-12-database-authorization-method 并重命名为 springsecurity-13-thymeleaf

15.2 添加 thymeleaf 依赖

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

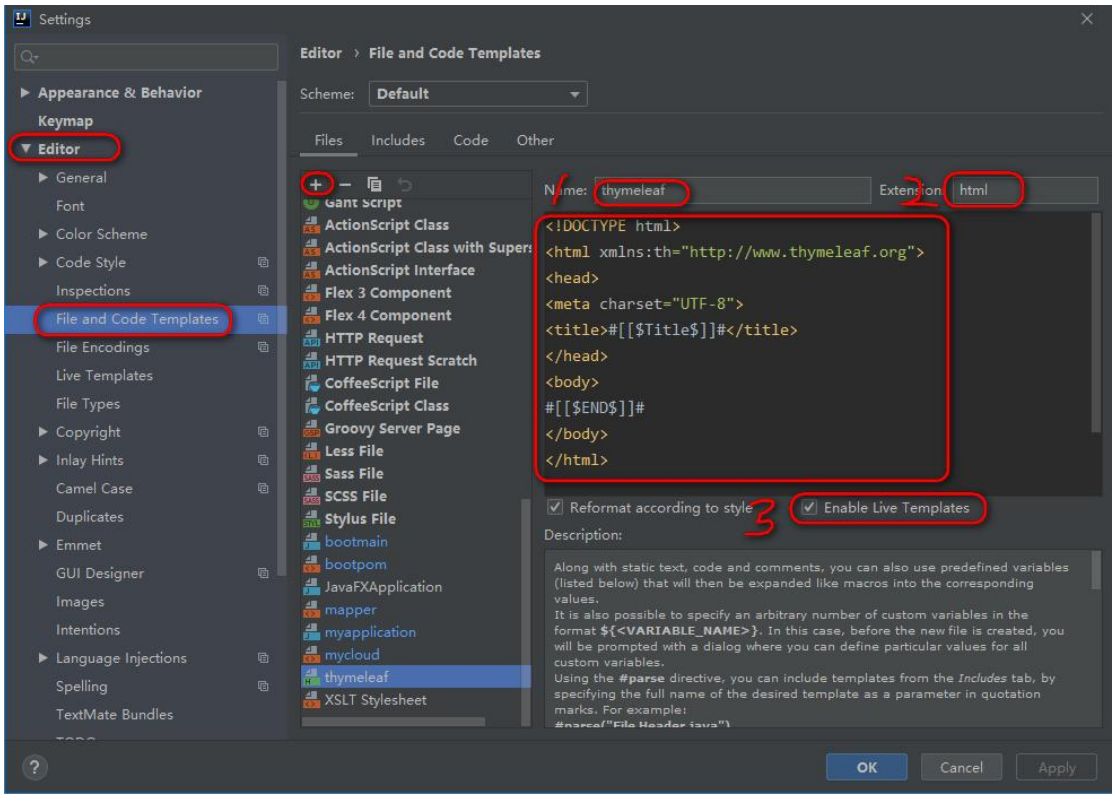
15.3 修改 application.yml

加入 thymeleaf 的配置

```
spring:
  thymeleaf:
    cache: false # 不使用缓存
    check-template: true  # 检查 thymeleaf 模板是否存在
```

15.4 idea 添加 thymeleaf 模板

【File】---》【Settings...】



模板名称 thymeleaf ，扩展名 html，具体内容如下：

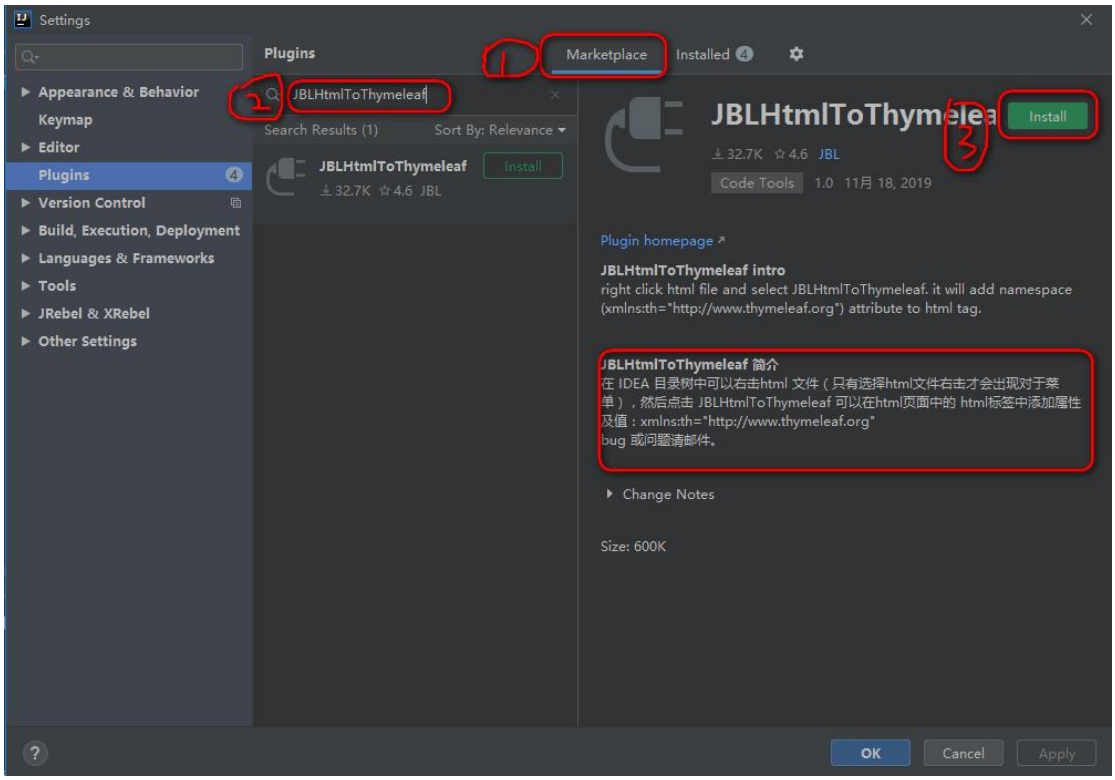
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>#[[$Title$]]#</title>
</head>
<body>
#[[$END$]]#
</body>
</html>
```

简要说明：

#[[$Title$]]# #[[END]]# 这两处的作用是，当你新建一个模板页面时，在<title>标签中输入标题内容后，只需要点击回车键，光标就会直接跳到<body>内，省去了你挪动鼠标，或者挪动方向键的步骤，也可以给你节省一点点时间。

15.5 idea 安装 html 转 thymeleaf 的插件（学员自行安装，不讲）

安装 JBLHtmlToThymeleaf 插件



15.6 新建 LoginController 和 IndexController

```java
@Controller
@RequestMapping("/login")
public class LoginController {
    /**
     * 跳转到登陆页面
     */
    @RequestMapping("/toLogin")
    public String toLogin(){
        return "login";
    }

}
```



```
@Controller
@RequestMapping("/index")
public class IndexController {


    /**
     * 登录成功后进入主页
     */
    @RequestMapping("/toIndex")
    public String toIndex(){
        return "main";
    }
}
```

15.7 创建静态页面 login.html 和 main.html

在 templates 下面创建 login.html 和 main.html

创建 login.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>用户登陆</title>
</head>
<body>
<h2>登录页面</h2>
<form action="/login/doLogin" method="post">
    <table>
        <tr>
            <td>用户名:</td>
            <td><input type="text" name="uname" value="thomas"></td>
```



```
        </tr>
        <tr>
            <td>密码:</td>
            <td><input type="password" name="pwd"></td>
            <span th:if="${param.error}">用户名或者密码错误</span>
        </tr>
        <tr>
            <td colspan="2">
                <button type="submit">登录</button>
            </td>
        </tr>
    </table>
</form>
</body>
```

创建 main.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>系统首页</title>
</head>
<body>
<h1 align="center">系统首页</h1>
<a href="/student/query">查询学生</a>
<br>
<a href="/student/add">添加学生</a>
<br>
<a href="/student/update">更新学生</a>
```



```
<br>
<a href="/student/delete">删除学生</a>
<br>
<a href="/student/export">导出学生</a>
<br>
<br><br><br>
<h2><a href="/logout">退出</a></h2>
<br>
</body>
</html>
```

15.8 修改安全配置文件 WebSecurityConfig

修改后如下：

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
//@Configuration
@Slf4j
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Bean
    public PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //设置登陆方式
        http.formLogin()//使用用户名和密码的登陆方式
                .usernameParameter("uname") //页面表单的用户名的 name
                .passwordParameter("pwd")//页面表单的密码的 name
                .loginPage("/login/toLogin") //自己定义登陆页面的地址
                .loginProcessingUrl("/login/doLogin")//配置登陆的 url
```



```
                .successForwardUrl("/index/toIndex") //登陆成功跳转的页
面
                .failureForwardUrl("/login/toLogin")//登陆失败跳转的页
面
                .permitAll();
        //配置退出方式
        http.logout()
                .logoutUrl("/logout")
                .logoutSuccessUrl("/login/toLogin")
                .permitAll();
        //配置路径拦截 的 url 的匹配规则
        http.authorizeRequests()
                //任何路径要求必须认证之后才能访问
                .anyRequest().authenticated();
        // 禁用 csrf 跨站请求攻击  后面可以使用 postman 工具测试，注意要禁用
csrf
        http.csrf().disable();
    }
}
```

15.9 修改 Studentcontroller

修改后如下：

```
@Controller
@Slf4j
@RequestMapping("/student")
public class StudentController {
    @GetMapping("/query")
    @PreAuthorize("hasAuthority('student:query')")
    public String queryInfo(){
        return "user/query";
    }
    @GetMapping("/add")
    @PreAuthorize("hasAuthority('student:add')")
```



```
    public String addInfo(){
        return "user/add";
    }
    @GetMapping("/update")
    @PreAuthorize("hasAuthority('student:update')")
    public String updateInfo(){
        return "user/update";
    }
    @GetMapping("/delete")
    @PreAuthorize("hasAuthority('student:delete')")
    public String deleteInfo(){
        return "user/delete";
    }
    @GetMapping("/export")
    @PreAuthorize("hasAuthority('student:export')")
    public String exportInfo(){
        return "/user/export";
    }
}
```

15.10 在 templates/user 下面创建学生管理的各个页面

创建 export.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org"
>
<head>
    <meta charset="UTF-8">
    <title>系统首页-学生管理</title>
</head>
<body>
<h1 align="center">系统首页-学生管理-导出</h1>
<a href="/index/toIndex">返回</a>
<br>
</body>
</html>
```


创建 query.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>系统首页-学生管理</title>
</head>
<body>
<h1 align="center">系统首页-学生管理-查询</h1>
<a href="/index/toIndex">返回</a>
<br>
</body>
</html>
```

创建 add.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>系统首页-学生管理</title>
</head>
<body>
<h1 align="center">系统首页-学生管理-新增</h1>
<a href="/index/toIndex">返回</a>
<br>
</body>
</html>
```

创建 update.html

```
<!DOCTYPE html>
<html lang="en">
```



```
<head>
    <meta charset="UTF-8">
    <title>系统首页-学生管理</title>
</head>
<body>
<h1 align="center">系统首页-学生管理-更新</h1>
<a href="/index/toIndex">返回</a>
<br>
</body>
</html>
```

创建 delete.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>系统首页-学生管理</title>
</head>
<body>
<h1 align="center">系统首页-学生管理-删除</h1>
<a href="/index/toIndex">返回</a>
<br>
</body>
</html>
```

15.11 创建 403 页面

在 static/error 下面创建 403.html

```
<!DOCTYPE html>
<html lang="en">
<head>
```



```
    <meta charset="UTF-8">
    <title>403</title>
</head>
<body>
<h2>403:你没有权限访问此页面</h2>
<a href="/index/toIndex">去首页</a>
</body>
</html>
```

15.12 启动测试

注意：如果出现 404 问题

```
 <parent>
        <artifactId>spring-boot-starter-parent</artifactId>
        <groupId>org.springframework.boot</groupId>
<!--        <version>2.3.12.RELEASE</version>-->
        <version>2.6.13</version>
        <!--修改了 sprigboot 的版本，然后再修改回去，就好了-->
        <relativePath/>
    </parent>
```

15.13 当用户没有某权限时，页面不展示该按钮（简单看下即可）

上一讲里面我们创建的项目里面是当用户点击页面上的链接请求到后台之后没有权限会跳转到 403，那么如果用户没有权限，对应的按钮就不显示出来，这样岂不是更好吗

我们接着上一个项目来改造

引入下面的依赖

```
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity5</artifactId>
</dependency>
```

修改 main.html 即可

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<head>
    <meta charset="UTF-8">
    <title>系统首页</title>
</head>
<body>
<h1 align="center">系统首页</h1>
<a href="/student/query"
sec:authorize="hasAuthority('student:query')" >查询用户</a>
<br>
<a href="/student/add" sec:authorize="hasAuthority('student:save')" >
添加用户</a>
<br>
<a href="/student/update"
sec:authorize="hasAuthority('student:update')" >更新用户</a>
<br>
<a href="/student/delete"
sec:authorize="hasAuthority('student:delete')" >删除用户</a>
<br>
<a href="/student/export"
sec:authorize="hasAuthority('student:export')" >导出用户</a>
<br>
<br><br><br>
<h2><a href="/logout">退出</a></h2>
<br>
</body>
</html>
```

重启启动登录后查看效果

16 springsecurity 集成图片验证码

以前因为我们自己写登陆的方法可以在自己的登陆方法里面去接收页面传过来的 code 再和 session 里面正确的 code 进行比较

16.1 概述

上一讲里面我们集成了 Thymeleaf 实现在页面链接的动态判断是否显示，那么在实际开发中，我们会遇到有验证码的功能，那么如何处理呢？

复制上一个工程 springsecurity-13-thymeleaf ,修改名字等 springsecurity-14-captcha

16.2 原理、存在问题、解决思路

我们知道 Spring Security 是通过过滤器链来完成了，所以它的解决方案是创建一个过滤器放到 Security 的过滤器链中，在自定义的过滤器中比较验证码

16.3 添加依赖（用于生成验证码）

```
<!--引入 hutool-->

<dependency>
```

```
    <groupId>cn.hutool</groupId>
    <artifactId>hutool-all</artifactId>
    <version>5.3.9</version>
</dependency>
```

16.4 添加一个获取验证码的接口

```
@Controller
@Slf4j
public class CaptchaController {
    @GetMapping("/code/image")
    public void getCaptcha(HttpServletRequest request,
HttpServletResponse response) throws IOException {
        //创建一个验证码
        CircleCaptcha circleCaptcha =
CaptchaUtil.createCircleCaptcha(200, 100, 2, 20);
        //放到 session 中
        // 为什么要重构？重构的快捷键是啥？
        String captchaCode=circleCaptcha.getCode();
        log.info("生成的验证码为：{}",captchaCode);

request.getSession().setAttribute("LOGIN_CAPTCHA_CODE",captchaCode);

ImageIO.write(circleCaptcha.getImage(),"JPEG",response.getOutputStre
am());
    }
}
```


16.5 创建验证码过滤器

```
@Component
@Slf4j
public class ValidateCodeFilter extends OncePerRequestFilter {
    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain) throws
ServletException, IOException {
        String requestURI = request.getRequestURI();
        log.info("请求的 URI 为: {}", requestURI);
        if (!requestURI.equals("/login/doLogin")) {
            doFilter(request, response, filterChain);
        } else {
            validateCode(request, response,filterChain);
        }
    }

    private void validateCode(HttpServletRequest request,
HttpServletResponse response,FilterChain filterChain) throws
IOException, ServletException {
        String enterCaptchaCode = request.getParameter("code");
        HttpSession session = request.getSession();
        String captchaCodeInSession = (String)
session.getAttribute("LOGIN_CAPTCHA_CODE");
        log.info("用户输入的验证码为: {},session 中的验证码为:
{}",enterCaptchaCode,captchaCodeInSession);
        //移除错误信息
        session.removeAttribute("captchaCodeErrorMsg");
        if (!StringUtils.hasText(captchaCodeInSession)) {
```



```
            session.removeAttribute("LOGIN_CAPTCHA_CODE");
        }
        if (!StringUtils.hasText(enterCaptchaCode)
|| !StringUtils.hasText(captchaCodeInSession)
|| !enterCaptchaCode.equalsIgnoreCase(captchaCodeInSession)) {
            //说明验证码不正确，返回登陆页面
            session.setAttribute("captchaCodeErrorMsg", "验证码不正确
");
          //重定向
            response.sendRedirect("/login/toLogin");
        }else{
            filterChain.doFilter(request,response);
        }
    }
}
```

16.6 修改 WebSecurityConfig（重点）

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
@Slf4j
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {



    @Resource
    private ValidateCodeFilter validateCodeFilter;

    @Override



    /**
```



```
    * Security 的http 请求配置
    *
    * @param http
    * @throws Exception
    */
   @Override
   protected void configure(HttpSecurity http) throws Exception {

       //设置登陆方式
       http.formLogin()//使用用户名和密码的登陆方式
               .usernameParameter("uname") //页面表单的用户名的 name
               .passwordParameter("pwd")//页面表单的密码的 name
               .loginPage("/login/toLogin") //自己定义登陆页面的地址
               .loginProcessingUrl("/login/doLogin")//配置登陆的 url
               .successForwardUrl("/index/toIndex") //登陆成功跳转的页
面
               .failureForwardUrl("/login/toLogin")//登陆失败跳转的页
面
               .permitAll();
       //配置退出方式
       http.logout()
               .logoutUrl("/logout")
               .logoutSuccessUrl("/login/toLogin")
               .permitAll();
       //配置路径拦截 的 url 的匹配规则

http.authorizeRequests().antMatchers("/code/image").permitAll()
               //任何路径要求必须认证之后才能访问
               .anyRequest().authenticated();
```



```
        // 禁用 csrf 跨站请求，注意不要写错了
        http.csrf().disable();
  // 配置登录之前添加一个验证码的过滤器
http.addFilterBefore(validateCodeFilter,UsernamePasswordAuthenticati
onFilter.class);
    }



    /**
     * 资源服务匹配放行【静态资源文件】
     *
     * @param web
     * @throws Exception
     */
   // @Override
    //public void configure(WebSecurity web) throws Exception {
      //  web.ignoring().mvcMatchers("/resources/**");
    //}



    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

16.7 修改 login.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
```



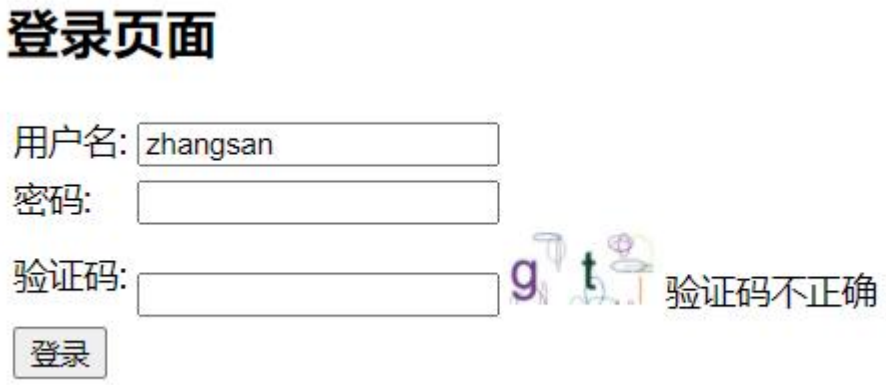
```
<head>
    <meta charset="UTF-8">
    <title>用户登陆</title>
</head>
<body>
<h2>登录页面</h2>
<!--${param.error}这个如果有值，就显示帐号或密码错误-->
<h4 th:if="${param.error}" style="color: #FF0000;">帐号或密码错误,请重新输入</h4>
<form action="/login/doLogin" method="post">
    <table>
        <tr>
            <td>用户名:</td>
            <td><input type="text" name="uname" value="zhangsan"></td>
        </tr>
        <tr>
            <td>密码:</td>
            <td><input type="password" name="pwd"></td>
        </tr>
        <tr>
            <td>验证码:</td>
            <td><input type="text" name="code"> <img src="/code/image" style="height:33px;cursor:pointer;" onclick="this.src=this.src">
                <span th:text="${session.captchaCodeErrorMsg}" style="color: #FF0000;" >username</span>
            </td>
        </tr>
        <tr>
            <td colspan="2">
```



```
            <button type="submit">登录</button>
        </td>
    </tr>
</table>
</form>
</body>
```

16.8 测试登录

故意输入错误验证码



17 Base64 和 JWT 学习

见《base64 及 jwt 学习文档.doc》

18 JWT+Spring Security+redis+mysql 实现认证

18.1 新建工程

复制工程 springsecurity-12-database-authorization-method，改名字为 springsecurity-16-jwt

注意这个工程已经有认证功能和基于方法授权的功能了。下面咱们看下如何设置 jwt 进行认证登录。

18.2 添加 jwt 依赖

```
<!-- 添加 jwt 的依赖 -->
<dependency>
    <groupId>com.auth0</groupId>
    <artifactId>java-jwt</artifactId>
    <version>3.11.0</version>
</dependency>
```

18.3 application.yml 中配置密钥

```
jwt:
  secretKey: mykey
```

18.4 jwt 功能类

com.powernode.util

```
@Component
@Slf4j
public class JwtUtils {
    //算法密钥
    @Value("${jwt.secretKey}")
    private String jwtSecretKey;

    /**
     * 创建 jwt
     *
     * @param userInfo 用户信息
     * @param authList 用户权限列表
     * @return 返回 jwt（JSON WEB TOKEN）
     */
    public String createToken(String userInfo, List<String> authList) {
        //创建时间
        Date currentTime = new Date();
        //过期时间，5 分钟后过期
        Date expireTime = new Date(currentTime.getTime() + (1000 * 60 *
5));
        //jwt 的 header 信息
        Map<String, Object> headerClaims = new HashMap<>();
        headerClaims.put("type", "JWT");
        headerClaims.put("alg", "HS256");
```



```
        //创建 jwt
        return JWT.create()
                .withHeader(headerClaims) // 头部
                .withIssuedAt(currentTime) //已注册声明：签发日期，发行日
期
                .withExpiresAt(expireTime) //已注册声明 过期时间
                .withIssuer("thomas")  //已注册声明，签发人
                .withClaim("userInfo", userInfo) //私有声明，可以自己定义
                .withClaim("authList", authList) //私有声明，可以自定义
                .sign(Algorithm.HMAC256(jwtSecretKey)); // 签名，使用
HS256 算法签名，并使用密钥
//        HS256 是一种对称算法，这意味着只有一个密钥，在双方之间共享。 使用
相同的密钥生成签名并对其进行验证。 应特别注意钥匙是否保密。
    }

    /**
     * 验证 jwt 的签名，简称验签
     *
     * @param token 需要验签的 jwt
     * @return 验签结果
     */
    public boolean verifyToken(String token) {
        //获取验签类对象
        JWTVerifier jwtVerifier =
JWT.require(Algorithm.HMAC256(jwtSecretKey)).build();
        try {
            //验签，如果不报错，则说明 jwt 是合法的，而且也没有过期
            DecodedJWT decodedJWT = jwtVerifier.verify(token);
            return true;
        } catch (JWTVerificationException e) {
            //如果报错说明 jwt 为非法的，或者已过期（已过期也属于非法的）
            log.error("验签失败：{}", token);
            e.printStackTrace();
        }
        return false;
    }

    /**
     * 获取用户 id
     *
     * @param token jwt
     * @return 用户 id
     */
    public String getUserInfo(String token) {
```



```
        //创建 jwt 验签对象
        JWTVerifier jwtVerifier =
JWT.require(Algorithm.HMAC256(jwtSecretKey)).build();
        try {
            //验签
            DecodedJWT decodedJWT = jwtVerifier.verify(token);
            //获取 payload 中 userInfo 的值，并返回
            return decodedJWT.getClaim("userInfo").asString();
        } catch (JWTVerificationException e) {
            e.printStackTrace();
        }
        return null;
    }

    /**
     * 获取用户权限
     *
     * @param token
     * @return
     */
    public List<String> getUserAuth(String token) {
        //创建 jwt 验签对象
        JWTVerifier jwtVerifier =
JWT.require(Algorithm.HMAC256(jwtSecretKey)).build();
        try {
            //验签
            DecodedJWT decodedJWT = jwtVerifier.verify(token);
            //获取 payload 中的自定义数据 authList（权限列表），并返回
            return
decodedJWT.getClaim("authList").asList(String.class);
        } catch (JWTVerificationException e) {
            e.printStackTrace();
        }
        return null;
    }

}
```

18.5 添加响应类

com.powernode.vo 包中

```
@Data
@AllArgsConstructor
@NoArgsConstructor
```

```
@Builder
public class HttpResult implements Serializable {
    private Integer code; //响应码
    private String msg; //响应消息
    private Object data; //响应对象
}
```

18.6 修改 SecurityUser 类

加入一个获取 SysUser 的方法

```
public SysUser getSysUser() {
    return sysUser;
}
```

18.7 新建认证成功处理器

```
/**
 * 认证成功处理器，当用户登录成功后，会执行此处理器
 */
@Component
public class MyAuthenticationSuccessHandler implements
AuthenticationSuccessHandler {
    //使用此工具类进行序列化
    @Resource
    private ObjectMapper objectMapper;
    @Resource
    private JwtUtils jwtUtils;

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request,
HttpServletResponse response, Authentication authentication) throws
IOException, ServletException {
        response.setCharacterEncoding("UTF-8");
        response.setContentType("text/html;charset=utf-8");
        //从认证对象中获取认证用户信息
        SecurityUser securityUser = (SecurityUser)
authentication.getPrincipal();
        String
userInfo=objectMapper.writeValueAsString(securityUser.getSysUser());
        List<SimpleGrantedAuthority> authorities =
(List<SimpleGrantedAuthority>) securityUser.getAuthorities();
```



```
        //TODO 这可以改成 lambda 表达式
        List<String> authList=new ArrayList<>();
        for (SimpleGrantedAuthority authority : authorities) {
            authList.add(authority.getAuthority());
        }
//使用 stream 流 1
List<String> test = authorities.stream().map(
        a -> {
           return a.getAuthority();
        }
).collect(Collectors.toList());
System.out.println("test = " + test);
//使用 stream 流 2
List<String> test111 =
authorities.stream().map(SimpleGrantedAuthority::getAuthority).colle
ct(Collectors.toList());
System.out.println("test111 = " + test111);

        // 创建 jwt
        String token = jwtUtils.createToken(userInfo,authList);

        //返回给前端 token
        HttpResult httpResult =
HttpResult.builder().code(200).msg("OK").data(token).build();
        PrintWriter writer = response.getWriter();
        writer.write(objectMapper.writeValueAsString(httpResult));
        writer.flush();
    }
}
```

18.8 新建 jwt 过滤器，用于检查 token 等

com.powernode.filter 包中新建类

```
/**
 * 定义一次性请求过滤器
 */
@Component
@Slf4j
public class JwtCheckFilter extends OncePerRequestFilter {
    @Resource
    private ObjectMapper objectMapper;
    @Resource
    private JwtUtils jwtUtils;
```

```
    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain) throws
ServletException, IOException {
        //获取请求 uri
        String requestURI = request.getRequestURI();
        // 如果是登录页面，放行
        if (requestURI.equals("/login")) {
            filterChain.doFilter(request, response);
            return;
        }
        //获取请求头中的 Authorization
        String authorization = request.getHeader("Authorization");
        //如果 Authorization 为空，那么不允许用户访问，直接返回
        if (!StringUtils.hasText(authorization)) {
            printFront(response, "没有登录！");
            return;
        }
        //Authorization 去掉头部的 Bearer 信息，获取 token 值
        String jwtToken = authorization.replace("Bearer ", "");
        //验签
        boolean verifyTokenResult = jwtUtils.verifyToken(jwtToken);
        //验签不成功
        if (!verifyTokenResult) {
            printFront(response, "jwtToken 已过期");
            return;
        }

        //从 payload 中获取 userInfo
        String userInfo = jwtUtils.getUserInfo(jwtToken);
        //从 payload 中获取授权列表
        List<String> userAuth = jwtUtils.getUserAuth(jwtToken);
        //创建登录用户
        SysUser sysUser = objectMapper.readValue(userInfo,
SysUser.class);
        SecurityUser securityUser = new SecurityUser(sysUser);
        //设置权限
        List<SimpleGrantedAuthority> authList =
userAuth.stream().map(SimpleGrantedAuthority::new).collect(Collector
s.toList());
        securityUser.setAuthorityList(authList);
```



```
        UsernamePasswordAuthenticationToken
usernamePasswordAuthenticationToke = new
UsernamePasswordAuthenticationToken(securityUser
                , null, authList);
        //通过安全上下文设置认证信息

SecurityContextHolder.getContext().setAuthentication(usernamePasswor
dAuthenticationToke);
        //继续访问相应的 rul 等
        filterChain.doFilter(request, response);

    }

    private void printFront(HttpServletResponse response, String
message) throws IOException {
        response.setCharacterEncoding("UTF-8");
        response.setContentType("application/json;charset=utf-8");
        PrintWriter writer = response.getWriter();
        HttpResult httpResult = new HttpResult();
        httpResult.setCode(-1);
        httpResult.setMsg(message);

        writer.print(objectMapper.writeValueAsString(httpResult));
        writer.flush();
    }
}
```

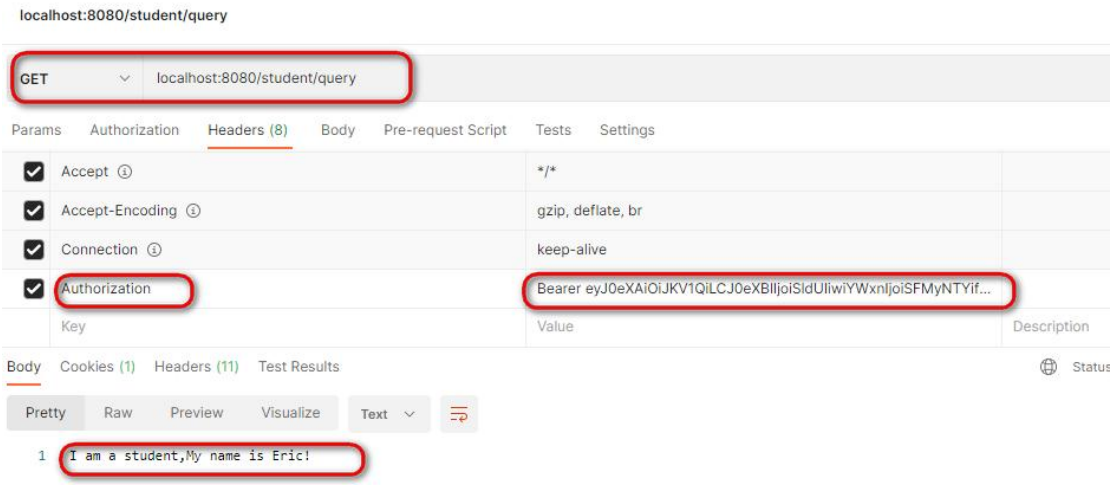
18.9 修改 web 安全配置类 WebSecurityConfig

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Resource
    private MyAuthenticationSuccessHandler
myAuthenticationSuccessHandler;
    @Resource
    private JwtCheckFilter jwtCheckFilter;
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.addFilterBefore(jwtCheckFilter,
UsernamePasswordAuthenticationFilter.class);

http.formLogin().successHandler(myAuthenticationSuccessHandler).perm
itAll();
```

```
http.authorizeRequests()
        .mvcMatchers("/student/**").hasAnyAuthority("student:query","
student:update")

                .anyRequest().authenticated(); //任何请求均需要认证（登录
成功）才能访问
        http.csrf().disable();
        //禁用 session

http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy
.STATELESS);
    }

    @Bean
    public PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }
}
```

18.10 启动测试

先登录系统，获取页面上返回的 token，然后使用 postman 在请求头中携带 token 发送请求即可。



18.11 测试后的问题

18.11.1 实现用户退出的问题

问题：因为 JWT 无状态，如果要实现退出功能无法实现。

解决办法:

使用 redis

步骤：

① 登陆成功之后把生成 JWT 存到 redis 中

key	value
logintoken:jwt	UsernamePasswordAuthenticationToken

② 用户退出时，从 redis 中删除该 token

③ 用户每次访问时，先校验 jwt 是否合法，如果合法再从 redis 里面取出 logintoken:jwt 判断这个 jwt 还存不存在，如果不存在就说是用户已经退出来，就返回未登陆。

18.11.2 启动 redis 并使用客户端工具连接到 redis

18.11.3 pom.xml 文件中加入 redis 依赖

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

18.11.4 配置 redis 信息

```
spring:
  redis:
    host: 192.168.43.33
    port: 6379
    database: 0
    password: 666666
```

18.11.5 修改认证成功处理器

添加依赖注入

```
@Resource
private StringRedisTemplate stringRedisTemplate;
```

代码中加入

```
stringRedisTemplate.opsForValue().set("logintoken:"+token,objectMapp
er.writeValueAsString(authentication),30, TimeUnit.MINUTES);
```


18.11.6 新建用户退出成功处理器

```
/**
 * 退出成功处理器，用户退出成功后，执行此处理器
 */
@Component
public class MyLogoutSuccessHandler implements LogoutSuccessHandler {
    //使用此工具类的对象进行序列化操作
    @Resource
    private ObjectMapper objectMapper;
    @Resource
    private StringRedisTemplate stringRedisTemplate;
    @Override
    public void onLogoutSuccess(HttpServletRequest request,
HttpServletResponse response, Authentication authentication) throws
IOException, ServletException {
        //从请求头中获取 Authorization 信息
        String authorization = request.getHeader("Authorization");
        //如果授权信息为空，返回前端
        if(null==authorization){
            response.setCharacterEncoding("UTF-8");

response.setContentType("application/json;charset=utf-8");
            HttpResult
httpResult=HttpResult.builder().code(-1).msg("token 不能为空
").build();
            PrintWriter writer = response.getWriter();

writer.write(objectMapper.writeValueAsString(httpResult));
            writer.flush();
            return;
        }
        //如果 Authorization 信息不为空，去掉头部的 Bearer 字符串
        String token = authorization.replace("Bearer ", "");
        //redis 中删除 token，这是关键点
        stringRedisTemplate.delete("logintoken:"+token);
        response.setCharacterEncoding("UTF-8");
        response.setContentType("application/json;charset=utf-8");
        HttpResult httpResult=HttpResult.builder().code(200).msg("退出
成功").build();
        PrintWriter writer = response.getWriter();
        writer.write(objectMapper.writeValueAsString(httpResult));
        writer.flush();
```



```
    }
}
```

配置用户成功退出处理器

修改 WebSecurityConfig

添加依赖

```
@Resource
private MyLogoutSuccessHandler myLogoutSuccessHandler;
```

添加代码

```
http.logout().logoutSuccessHandler(myLogoutSuccessHandler);
http.csrf().disable(); //禁用跨域请求保护 要不然 logout 不能访问
```

18.11.7 修改 jwtcheckfilter

添加依赖注入

```
@Resource
private StringRedisTemplate stringRedisTemplate;
```

代码中加入

```
//        从 redis 中获取 token
        String tokenInRedis =
stringRedisTemplate.opsForValue().get("logintoken:" + jwtToken);
        if(!StringUtils.hasText(tokenInRedis)){
            printFront(response, "用户已退出，请重新登录");
            return;
        }
```

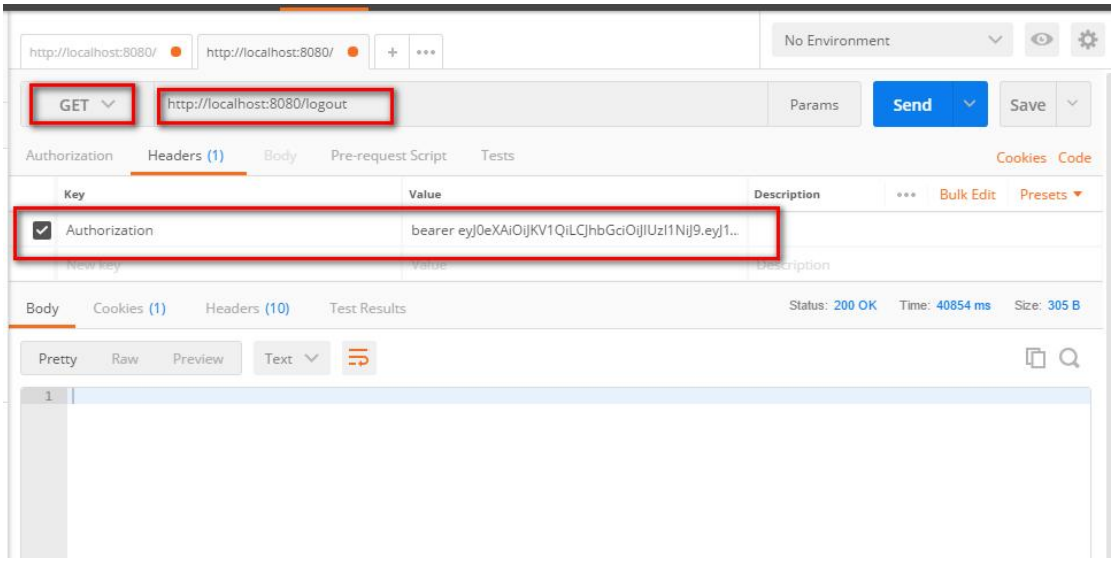
18.11.8 启动程序并登录测试

登录后查看 redis 中是否存储了 token

18.11.9 使用 token 访问/student/query

使用 postman 测试，发现可以正常访问

18.11.10 使用 postman 退出系统



注意携带 token，才能退出啊

注意：要禁用跨域请求保护，要不然使用 postman 无法访问

```
http.csrf().disable(); //禁用跨域请求保护
```

18.11.11 再次使用 token 访问/student/query

现象：发现已经不能正常访问了

原因：虽然 token 本身并没有过期，但是 redis 中已经删除了该 token，所以不能正常访问了

