

基于Element的前端管理端与Java后端的联调(下)

1、前端管理端增加登录页面；

参考布局组件[组件](#)|[Element](#)，使用header+main组件：

```
1 <el-container>
2   <el-header>Header</el-header>
3   <el-main>Main</el-main>
4 </el-container>
```

Header组件主要是title:

```
1 <h1>
2   Quiz后台管理系统
3 </h1>
```

Main布局中的登录具体内容可以借助Github Copilot，替你生成：

```
1 <template>
2   <el-container>
3     <el-header style="font-size: 40px; background-color: rgb(238, 241,
4       246); display: flex; justify-content: center; align-items: center;">Quiz后台管
5       理系统登录</el-header>
6     <el-main>
7       <el-form :model="form" :rules="rules" ref="loginForm" label-
8         width="80px" class="login-form">
9         <el-form-item label="用户名" prop="username">
10           <el-input v-model="form.username" placeholder="请输入用户名"></el-
11             input>
12           </el-form-item>
13           <el-form-item label="密码" prop="password">
14             <el-input v-model="form.password" type="password" placeholder="请
15               输入密码"></el-input>
16           </el-form-item>
17           <el-form-item>
18             <el-button type="primary" @click="onSubmit">登录</el-button>
19           </el-form-item>
20         </el-form>
21       </el-main>
22     </el-container>
23   </template>
24
25   <script>
26     import axios from "axios";
27
28     export default {
29       name: 'LoginView',
30       data() {
31         return {
32           form: {
33             username: '',
34             password: ''
```

```
30     },
31     rules: {
32       username: [{ required: true, message: '请输入用户名', trigger: 'blur' }],
33       password: [{ required: true, message: '请输入密码', trigger: 'blur' }],
34     },
35   },
36 },
37 methods: {
38   onSubmit() {
39     this.$refs.loginForm.validate((valid) => {
40       if (valid) {
41         console.log('提交表单:', this.form);
42         // 调用后端登录接口
43         this.login();
44       } else {
45         console.error('表单验证失败');
46       }
47     });
48   },
49
50   login() {
51     axios.post('login', {
52       username: this.form.username,
53       password: this.form.password
54     }, {
55       headers: {
56         'Content-Type': 'application/json'
57       }
58     })
59     .then(res => {
60       console.log('登录响应:', res.data);
61       if (res.data.code === 1) {
62         //const token = res.data.data;
63         //localStorage.setItem('jwt_token', token); // 保存 token
64         this.$router.push('/user'); // 登录成功跳转
65       } else {
66         this.$message.error('登录失败: ' + res.data.message);
67       }
68     })
69     .catch(err => {
70       console.error(err);
71       this.$message.error('登录异常');
72     });
73   }
74 },
75 };
76 </script>
77
78 <style>
79 .login-form {
80   max-width: 400px;
81   margin: 50px auto;
82 }
83 </style>
```

router路由中，添加LoginView.vue路由：

```
1 import Vue from 'vue'
2 import VueRouter from 'vue-router'
3 import Userview from '../views/elment/Userview.vue'
4 import Questionview from '@/views/elment/Questionview.vue'
5 import Loginview from '@/views/elment/Loginview.vue'
6
7 Vue.use(VueRouter)
8 const routes = [
9   {
10     path: '/',
11     redirect: '/login'
12   },
13   {
14     path: '/user',
15     name: 'user',
16     component: Userview
17   },
18   {
19     path: '/question',
20     name: 'question',
21     component: Questionview
22   },
23   {
24     path: '/login',
25     name: 'login',
26     component: Loginview
27   }
28 ]
29 const router = new VueRouter({
30   routes
31 })
32 export default router
```

2、Java后端增加登录/login接口；

Java后台在前端成功login后，返回前端的Result对象：

```
1 {
2   "code":1,
3   "msg":"login success",
4   "data":后端生成的jwt token
5 }
```

实体类复用User.java；

Controller实现：

在UserController中添加如下的代码：

```
1 @PostMapping("/login")
2 public Result login(@RequestBody Map<String, String> loginData){
```

```

3     String username = loginData.get("username");
4     String password = loginData.get("password");
5
6     if (StringUtils.isAnyBlank(username, password)) {
7         return Result.error("用户名或密码为空");
8     }
9     User userResult = userService.login(username, password);
10    if(userResult!=null){
11        return Result.success("用户登录成功");
12    }else{
13        return Result.error("用户登录失败");
14    }
15}

```

此处也可以不用Map<String, String> loginData接收数据，而是采用DAO实体类：

```

1 public class LoginRequest {
2     private String username;
3     private String password;
4
5     // Getter 和 Setter 方法
6     public String getUsername() {
7         return username;
8     }
9
10    public void setUsername(String username) {
11        this.username = username;
12    }
13
14    public String getPassword() {
15        return password;
16    }
17
18    public void setPassword(String password) {
19        this.password = password;
20    }
21}

```

Service实现：

在Service层中，需要对密码进行加密后再发送到mapper层进行查询；

```

1 //接口:
2 public User login(String username, String password);
3
4 //实现:
5 public User login(String username, String password){
6     //对密码进行加密;
7     final String SALT = "com.quiz";
8     String encrptedPassword = DigestUtils.md5DigestAsHex((SALT +
9     password).getBytes());
10
11    return userMapper.getByNameAndPassword(username, encrptedPassword);
11}

```

Mapper层实现：

```
1 |     @Select("select * from user where userName=#{username} AND userPassword=#{password}")
2 |     public User getByNameAndPassword(String username, String password)
```

3、登录认证JWT令牌

[Day12-02. 登录校验-概述哔哩哔哩bilibili](#)

```
1 | https://www.jwt.io/
```

JWT是一种简介、自包含的json格式数字签名，包含三部分：

- 第一部分，Header（头）；
- 第二部分，Payload(有效载荷)；
- 第三部分，Signature(签名)；

JavaWeb中要使用JWT令牌，我们需要在pom.xml中添加如下的依赖：

```
1 | <dependency>
2 |     <groupId>io.jsonwebtoken</groupId>
3 |     <artifactId>jjwt-api</artifactId>
4 |     <version>0.11.5</version>
5 | </dependency>
6 | <dependency>
7 |     <groupId>io.jsonwebtoken</groupId>
8 |     <artifactId>jjwt-impl</artifactId>
9 |     <version>0.11.5</version>
10 |    <scope>runtime</scope>
11 | </dependency>
12 | <dependency>
13 |     <groupId>io.jsonwebtoken</groupId>
14 |     <artifactId>jjwt-jackson</artifactId> <!-- 使用 Jackson 解析 JSON -->
15 |     <version>0.11.5</version>
16 |     <scope>runtime</scope>
17 | </dependency>
```

生成/解析令牌的工具类：

```
1 | import io.jsonwebtoken.Jwts;
2 | import io.jsonwebtoken.SignatureAlgorithm;
3 | import io.jsonwebtoken.security.Keys;
4 |
5 | import java.security.Key;
6 | import java.util.Date;
7 |
8 | public class JwtUtil {
9 |
10 |     // 生成密钥（实际项目中要保存在配置中）
11 |     private static final Key SECRET_KEY =
12 |         Keys.hmacShaKeyFor("mySuperSecretKey1234567890abcdef".getBytes());
13 |     // 生成 JWT 字符串
```

```

14     public static String generateToken(String username) {
15         long expirationMillis = 1000 * 60 * 60; // 1 小时
16         return Jwts.builder()
17             .setSubject(username)
18             .setIssuedAt(new Date())
19             .setExpiration(new Date(System.currentTimeMillis() +
expirationMillis))
20             .signWith(SECRET_KEY, SignatureAlgorithm.HS256)
21             .compact();
22     }
23
24     // 解析 JWT 并获取用户名
25     public static String parseToken(String token) {
26         return Jwts.parserBuilder()
27             .setSigningKey(SECRET_KEY)
28             .build()
29             .parseClaimsJws(token)
30             .getBody()
31             .getSubject();
32     }
33 }
```

测试工具类的代码：

```

1     @Test
2     public void testJWT(){
3         //生成Toekn
4         String token = JwtUtil.generateToken("toms");
5         System.out.println("generated token:" + token);
6
7         //解析Toekn
8         String username = JwtUtil.parseToken(token);
9         System.out.println("解析出的用户名: " + username);
10    }
```

上述代码的输出：

```

OpenJDK 64-Bit Server VM Warning: Sharing is only supported for boot loader classes because bootstrap classpath
generated token:eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0b21zIiwiaWF0IjoxNzU0MTIwNjQwLCJleHAiOjE3NTQzMjQyNDB9.SomLOLDX13
解析出的用户名: toms
```

上述的工具类只能就一个username生成密钥，那如何对多个用户属性，比如id, username, email等生成密钥呢？

```

1 import io.jsonwebtoken.Jwts;
2 import io.jsonwebtoken.SignatureAlgorithm;
3 import io.jsonwebtoken.security.Keys;
4
5 import java.security.Key;
6 import java.util.Date;
7
8 public class JwtUtil {
9
10     // 生成密钥（实际项目中要保存在配置中）
```

```

11     private static final Key SECRET_KEY =
12         Keys.hmacShaKeyFor("mySuperSecretKey1234567890abcdef".getBytes());
13
14     // Token 有效期: 1小时 (可根据需要调整)
15     private static final long EXPIRATION_TIME_MS = 60 * 60 * 1000;
16
17     // 生成 JWT 字符串
18     public static String generateTokenWithClaims(Claims claim) {
19         return Jwts.builder()
20             .setClaims(claim)
21             .setIssuedAt(new Date())
22             .setExpiration(new Date(System.currentTimeMillis() +
EXPIRATION_TIME_MS))
23             .signWith(SECRET_KEY, SignatureAlgorithm.HS256)
24             .compact();
25
26
27     // 解析 JWT
28     public static Claims parseTokenReturnClaims(String token) {
29         return Jwts.parserBuilder()
30             .setSigningKey(SECRET_KEY)
31             .build()
32             .parseClaimsJws(token)
33             .getBody();
34     }

```

相应的测试代码：

```

1 public class JwtTest {
2     public static void main(String[] args) {
3         //生成JWT令牌;
4         Claims userClaims = Jwts.claims(); // 新建一个 Claims 对象
5         claims.put("id", "123");
6         claims.put("username", "toms");
7         String token = JwtUtil.generateToken(userClaims);
8         System.out.println("token:" + token);
9         //解析JWT令牌;
10        Claims claim = JwtUtil.parseToken(token);
11        System.out.println("== 解析后的 JWT Claims ==");
12        System.out.println("ID: " + claims.get("id", String.class));
13        System.out.println("Username: " + claims.get("username",
String.class));
14    }
15 }

```

4、基于Filter实现登录验证

登录验证的主要流程

- 前端登录成功时，后台生成Jwt令牌并发送给前端；
- 前端设置Jwt令牌，后续所有的访问都带上令牌；
- 后台对Jwt令牌验证过滤；

后端登录成功生成Jwt令牌：

Controller中的代码：

```
1 @PostMapping("/login")
2 public Result login(@RequestBody Map<String, String> loginData){
3
4     String username = loginData.get("username");
5     String password = loginData.get("password");
6
7     if (StringUtils.isAnyBlank(username, password)) {
8         return Result.error("用户名或密码为空");
9     }
10    User userResult = userService.login(username, password);
11    if(userResult!=null){
12        Claims claims = Jwts.claims();
13        claims.put("id", userResult.getId());
14        claims.put("username", userResult.getUserName());
15
16        String token = JwtUtil.generateTokenWithClaims(claims);
17        Result result = Result.success("用户登录成功");
18        result.setData(token);
19        return result;
20    }else{
21        return Result.error("用户登录失败");
22    }
23}
24
```

前端测试后，可以获取Jwt Token：

The screenshot shows the Network tab of a browser's developer tools. A successful POST request to the 'login' endpoint is highlighted. The response body is visible, containing a JSON object with 'code': 1, 'msg': '用户登录成功', and a large 'data' field containing a JWT token.

序号id	添加日期	用户名	操作
3	2025-07-22T23:40:50.000+00:00	testuser	<button>编辑</button> <button>删除</button>
4	2025-07-23T00:44:53.000+00:00	john	<button>编辑</button> <button>删除</button>
5	2025-07-25T00:52:25.000+00:00	toms	<button>编辑</button> <button>删除</button>
6	2025-07-24T00:49:07.000+00:00	fromapifox	<button>编辑</button> <button>删除</button>
7	2025-07-24T00:50:26.000+00:00	fox	<button>编辑</button> <button>删除</button>

前端存储Jwt令牌到本地

```
1 login() {
2     axios.post('login', {
3         username: this.form.username,
4         password: this.form.password
5     }, {
6         headers: {
```

```

7         'Content-Type': 'application/json'
8     }
9 })
10    .then(res => {
11      console.log('登录响应:', res.data);
12      if (res.data.code === 1) {
13        const token = res.data.data;
14        localStorage.setItem('jwt_token', token); // 将jwt_token保存到本地;
15        this.$router.push('/user'); // 登录成功跳转
16      } else {
17        this.$message.error('登录失败: ' + res.data.message);
18      }
19    })
20    .catch(err => {
21      console.error(err);
22      this.$message.error('登录异常');
23    });
24  }
25 },

```

如何保证每次前端向后端发送数据请求时，都带上jwt_token，需要在前端main.js中，设置全局拦截器：

```

1 import axios from 'axios';
2
3 // 添加请求拦截器
4 axios.interceptors.request.use(
5   (config) => {
6     // 从localStorage 中获取 token
7     const token = localStorage.getItem('jwt_token');
8     if (token) {
9       // 在请求头中添加 token 字段
10      config.headers.token = token;
11    }
12    return config;
13  },
14  (error) => {
15    // 请求错误处理
16    return Promise.reject(error);
17  }
18);

```

后端拦截类Filter：

新建filter目录，创建JwtFilter类作为Filter的实现类：

```

1 package com.tfzhang.quiz.filter;
2
3 import jakarta.servlet.*;
4 import jakarta.servlet.annotation.WebFilter;
5
6 import java.io.IOException;
7
8 @WebFilter(urlPatterns = "/*")
9 public class JwtFilter implements Filter {
10

```

```

11     //初始化，只调用一次;
12     public void init(FilterConfig filterConfig) throws ServletException {
13         System.out.println("JwtFilter init");
14     }
15
16     @Override //拦截请求，每当有新请求时;
17     public void doFilter(ServletRequest req, ServletResponse res,
18     FilterChain chain) throws IOException, ServletException {
19         System.out.println("JwtFilter拦截到请求");
20         chain.doFilter(req, res);
21     }
22
23     //销毁，只调用一次;
24     public void destroy() {
25         System.out.println("JwtFilter destroy");
26     }
27

```

登录验证的业务逻辑代码书写在doFilter()方法中，当前对所有的请求放行：

```

1 //拦截所有类型的请求;
2 @WebFilter(urlPatterns = "/*")
3
4 //对请求进行放行;
5 chain.doFilter(req, res);

```

要使得JwtFilter发挥作用，还需要在启动类中添加新标签：

```

1 @ServletComponentScan //使得Filter发挥作用;
2 @SpringBootApplication
3 public class QuizApplication {
4
5     public static void main(String[] args) {
6         SpringApplication.run(QuizApplication.class, args);
7     }
8
9 }

```

后端登录验证

后端登录验证的思路：

- 1、拦截所有请求，辨识请求类型，如果是/login请求，放行；
- 2、如果非/login请求，获取请求头的Jwt令牌；
- 3、如果Jwt令牌存在，解析成功则放行；如果解析不成功，则返回登录不成功信息；
- 4、如果Jwt令牌不存在，则返回登录不成功信息；

对应的逻辑步骤：

```

1 package com.tfzhang.quiz.filter;
2
3 import com.alibaba.fastjson.JSONObject;
4 import com.tfzhang.quiz.model.Result;

```

```
5 import jakarta.servlet.*;
6 import jakarta.servlet.annotation.webFilter;
7 import jakarta.servlet.http.HttpServletRequest;
8 import jakarta.servlet.http.HttpServletResponse;
9 import org.springframework.util.StringUtils;
10
11 import java.io.IOException;
12
13 import com.tfzhang.quiz.utils.JwtUtil;
14
15 @WebFilter(urlPatterns = "/*")
16 public class JwtFilter implements Filter {
17
18     public void init(FilterConfig filterConfig) throws ServletException {
19         System.out.println("JwtFilter init");
20     }
21
22     @Override
23     public void doFilter(ServletRequest req, ServletResponse res,
24     FilterChain chain) throws IOException, ServletException {
25         System.out.println("JwtFilter拦截到请求");
26         //    chain.doFilter(req, res);
27
28         HttpServletRequest request = (HttpServletRequest) req;
29         HttpServletResponse response = (HttpServletResponse) res;
30
31         //1、获取请求url;
32         String url=request.getRequestURL().toString();
33         //2、判断url中是否包含login,如果包含，则说明是登录操作，放行;
34         if(url.contains("login")){
35             chain.doFilter(request, response);
36             return;
37         }
38
39         //3、获取请求头中的令牌(token)
40         String token = request.getHeader("token");
41
42         //4、判断令牌是否存在，如果不存在，返回未登录信息。
43         if(!StringUtils.hasLength(token)){
44             response.setStatus(HttpServletResponse.SC_UNAUTHORIZED); // 设置
45             //状态码为 401
46             Result result = Result.error("NOT_LOGIN");
47             //手动将对象转为json，并传回前端;
48             String noLogin= JSONObject.toJSONString(result);
49             response.setContentType("application/json;charset=UTF-8");
50             res.getWriter().write(noLogin);
51             return;
52         }
53
54         //5、解析token，如果解析失败，返回未登录信息。
55         try {
56             JwtUtil.parseToken(token);//只要解析不成功，就说明有问题;
57         }catch(Exception e){
58             e.printStackTrace();
59             response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
60             Result result = Result.error("NOT_LOGIN");
61             //手动将对象转为json，并传回前端;
62             String noLogin=JSONObject.toJSONString(result);
63         }
64     }
65 }
```

```

61         response.setContentType("application/json; charset=UTF-8");
62         res.getWriter().write(noLogin);
63         return;
64     }
65
66     //6、放行。
67     chain.doFilter(req, res);
68 }
69
70 public void destroy() {
71     System.out.println("JwtFilter destroy");
72 }
73 }
```

其中的JSONObject来自于：

```

1 <dependency>
2   <groupId>com.alibaba</groupId>
3   <artifactId>fastjson</artifactId>
4   <version>1.2.83</version>
5 </dependency>
```

5、解决访问/user页面渲染问题

在 Vue Router 配置导航守卫：

在 Vue Router 的全局导航守卫中检查登录状态。如果没有 `jwt_token`，在进入受保护的页面（如 `/user` 或 `/question`）之前直接跳转到 `/login`。修改 `router/index.js` 文件：

```

1 import vue from 'vue'
2 import VueRouter from 'vue-router'
3 import Userview from '../views/element/Userview.vue'
4 import Questionview from '@/views/element/Questionview.vue'
5 import Loginview from '@/views/element/Loginview.vue'
6 // import { meta } from '@babel/eslint-parser'
7
8 Vue.use(VueRouter)
9
10 const routes = [
11   {
12     path: '/',
13     redirect: '/login'
14   },
15   {
16     path: '/user',
17     name: 'user',
18     component: Userview,
19     meta: {
20       requiresAuth: true // 需要登录才能访问
21     }
22   },
23   {
24     path: '/question',
25     name: 'question',
26     component: Questionview,
```

```

27     meta: {
28       requiresAuth: true // 需要登录才能访问
29     }
30   },
31   {
32     path: '/login',
33     name: 'Login',
34     component: LoginView
35   }
36 ]
37
38 const router = new VueRouter({
39   routes
40 })
41
42 // 全局导航守卫
43 router.beforeEach((to, from, next) => {
44   // 检查路由是否需要登录
45   if (to.matched.some(record => record.meta.requiresAuth)) {
46     const token = localStorage.getItem('jwt_token'); // 从 localStorage 获取 token
47     if (!token) {
48       // 如果没有 token, 跳转到登录页面
49       next('/login');
50     } else {
51       // 如果有 token, 继续访问目标页面
52       next();
53     }
54   } else {
55     // 如果路由不需要登录, 直接继续
56     next();
57   }
58 });
59 export default router

```

6、添加前端退出功能

添加按钮组件：

```

1 <el-header style="font-size: 40px; background-color: rgb(238, 241, 246)"
2   >Quiz后台管理
3     <el-button type="danger" size="mini" @click="logout">退出</el-button>
4   </el-header>

```

js脚本中添加logout()方法：

```

1  logout() {
2      // 删除 token
3      localStorage.removeItem('jwt_token');
4      // 跳转到登录页面
5      this.$router.push('/login');
6      this.$message({
7          type: 'success',
8          message: '已退出登录',
9      });
10 }

```

7、跨域访问：

要配置对应的类：

```

1 package com.tfzhang.quiz.config;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.web.servlet.config.annotation.CorsRegistry;
6 import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
7
8 @Configuration
9 public class CorsConfig implements WebMvcConfigurer {
10     @Override
11     public void addCorsMappings(CorsRegistry registry) {
12         registry.addMapping("/**")
13             .allowedOrigins("http://localhost:8081")
14             .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
15             .allowedHeaders("*")
16             .allowCredentials(true);
17     }
18 }

```

注意：.allowedMethods一定要包含OPTIONS，虽然我们的前端代码里没有显示地使用，但是跨域访问时候，会隐式地用到。

前端原来的代理服务关掉(注释掉)：

```

1 // proxy: {
2 //   '/': {
3 //     target: 'http://localhost:8080',
4 //     changeOrigin: true,
5 //     ws: true,
6 //     pathRewrite: {
7 //       '^/': ''
8 //     }
9 //   }
10 //}

```

对于axios的设置：

```
1 | axios.get('/login', {
2 |   withCredentials: true
3 | });
4 | //或者统一配置;
5 | axios.defaults.withCredentials = true;
```

main.js设置前端的baseUrl:

```
1 | // 设置 Axios 的全局 baseURL
2 | axios.defaults.baseURL = 'http://localhost:8080';
```

chatgpt的地址:

```
1 | https://chatgpt.com/c/6893641d-05e8-8330-af86-313d8dcf690a
```