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：

```
<el-container>
  <el-header>Header</el-header>
  <el-main>Main</el-main>
</el-container>
```

Header组件主要是title:

```
<h1>
    Quiz后台管理系统
</h1>
```

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

```
<template>
  <el-container>
    <el-header style="font-size: 40px; background-color: rgb(238, 241, 246);display: flex; justify-content: center; align-items: center;">Quiz后台管理系统登录</el-header>
    <el-main>
      <el-form :model="form" :rules="rules" ref="loginForm" label-width="80px" class="login-form">
        <el-form-item label="用户名" prop="username">
          <el-input v-model="form.username" placeholder="请输入用户名"></el-input>
        </el-form-item>
        <el-form-item label="密码" prop="password">
          <el-input v-model="form.password" type="password" placeholder="请输入密码"></el-input>
        </el-form-item>
        <el-form-item>
          <el-button type="primary" @click="onSubmit">登录</el-button>
        </el-form-item>
      </el-form>
    </el-main>
  </el-container>
</template>

<script>
import axios from "axios";

export default {
  name: 'LoginView',
  data() {
    return {
      form: {
        username: '',
        password: '',
```

```
      },
      rules: {
        username: [{ required: true, message: '请输入用户名', trigger: 'blur' }],
        password: [{ required: true, message: '请输入密码', trigger: 'blur' }],
      },
    };
  },
  methods: {
    onSubmit() {
      this.$refs.loginForm.validate((valid) => {
        if (valid) {
          console.log('提交表单:', this.form);
          // 调用后端登录接口
          this.login();
        } else {
          console.error('表单验证失败');
        }
      });
    },

    login() {
      axios.post('login', {
        username: this.form.username,
        password: this.form.password
      }, {
        headers: {
          'Content-Type': 'application/json'
        }
      })
      .then(res => {
        console.log('登录响应:', res.data);
        if (res.data.code === 1) {
          //const token = res.data.data;
          //localStorage.setItem('jwt_token', token); // 保存 token
          this.$router.push('/user'); // 登录成功跳转
        } else {
          this.$message.error('登录失败：' + res.data.message);
        }
      })
      .catch(err => {
        console.error(err);
        this.$message.error('登录异常');
      });
    }
  },
};
</script>

<style>
.login-form {
  max-width: 400px;
  margin: 50px auto;
}
</style>
```

router路由中，添加LoginView.vue路由：

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import UserView from '../views/elment/UserView.vue'
import QuestionView from '@/views/elment/QuestionView.vue'
import LoginView from '@/views/elment/LoginView.vue'

Vue.use(VueRouter)
const routes = [
  {
    path: '/',
    redirect: '/login'
  },
  {
    path:'/user',
    name: 'user',
    component: UserView
  },
  {
    path:'/question',
    name:'question',
    component:QuestionView
  },
  {
    path:'/login',
    name:'login',
    component: LoginView
  }
]
const router = new VueRouter({
  routes
})
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：

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

实体类复用User.java；

Controller实现：

在UserController中添加如下的代码：

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

```
            String username = loginData.get("username");
            String password = loginData.get("password");

            if (StringUtils.isAnyBlank(username, password)) {
                return Result.error("用户名或密码为空");
            }
            User userResult = userService.login(username, password);
            if(userResult!=null){
                return Result.success("用户登录成功");
            }else{
                return Result.error("用户登录失败");
            }
        }
```

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

```
public class LoginRequest {
    private String username;
    private String password;

    // Getter 和 Setter 方法
    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

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；

```
//接口：
public User login(String username, String password);

//实现：
public User login(String username, String password){
        //对密码进行加密；
        final String SALT = "com.quiz";
        String encrptedPassword = DigestUtils.md5DigestAsHex((SALT +
password).getBytes());

        return userMapper.getByNameAndPassword(username,encrptedPassword);
}
```

Mapper层实现：

```
    @Select("select * from user where userName=#{username} AND userPassword=#{password}")
    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]

```
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：

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.11.5</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.11.5</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId> <!-- 使用 Jackson 解析 JSON -->
    <version>0.11.5</version>
    <scope>runtime</scope>
</dependency>
```

生成/解析令牌的工具类：

```
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.security.Keys;

import java.security.Key;
import java.util.Date;

public class JwtUtil {

    // 生成密钥（实际项目中要保存在配置中）
    private static final Key SECRET_KEY = Keys.hmacShaKeyFor("mySuperSecretKey1234567890abcdef".getBytes());

    // 生成 JWT 字符串
```

```
    public static String generateToken(String username) {
        long expirationMillis = 1000 * 60 * 60; // 1 小时
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() +
expirationMillis))
                .signWith(SECRET_KEY, SignatureAlgorithm.HS256)
                .compact();
    }

    // 解析 JWT 并获取用户名
    public static String parseToken(String token) {
        return Jwts.parserBuilder()
                .setSigningKey(SECRET_KEY)
                .build()
                .parseClaimsJws(token)
                .getBody()
                .getSubject();
    }
}
```

测试工具类的代码：

```
    @Test
    public void testJWT(){
        //生成Toekn
        String token = JwtUtil.generateToken("toms");
        System.out.println("generated token:"+token);

        //解析Toekn
        String username = JwtUtil.parseToken(token);
        System.out.println("解析出的用户名：" + username);

    }
```

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.eyJzdWIiOiJ0b21zIiwiaWF0IjoxNzU0MTIwNjQwLCJleHAiOjE3NTQxMjQyNDB9.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？

```
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.security.Keys;

import java.security.Key;
import java.util.Date;

public class JwtUtil {

    // 生成密钥（实际项目中要保存在配置中）
```

```
    private static final Key SECRET_KEY = Keys.hmacShaKeyFor("mySuperSecretKey1234567890abcdef".getBytes());

    // Token 有效期：1小时（可根据需要调整）
    private static final long EXPIRATION_TIME_MS = 60 * 60 * 1000;

    // 生成 JWT 字符串
    public static String generateTokenWithClaims(Claims claim) {
        return Jwts.builder()
                .setClaims(claim)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + EXPIRATION_TIME_MS))
                .signWith(SECRET_KEY, SignatureAlgorithm.HS256)
                .compact();
    }


    // 解析 JWT
    public static Claims parseTokenReturnClaims(String token) {
        return Jwts.parserBuilder()
                .setSigningKey(SECRET_KEY)
                .build()
                .parseClaimsJws(token)
                .getBody();
    }
```

相应的测试代码：

```
public class JwtTest {
    public static void main(String[] args) {
        //生成JWT令牌;
        Claims userClaims = Jwts.claims(); // 新建一个 Claims 对象
        claims.put("id", "123");
        claims.put("username", "toms");
        String token = JwtUtil.generateToken(userClaims);
        System.out.println("token:"+token);
        //解析JWT令牌;
        Claims claim = JwtUtil.parseToken(token);
        System.out.println("=== 解析后的 JWT Claims ===");
        System.out.println("ID: " + claims.get("id", String.class));
        System.out.println("Username: " + claims.get("username", String.class));
    }
}
```

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；

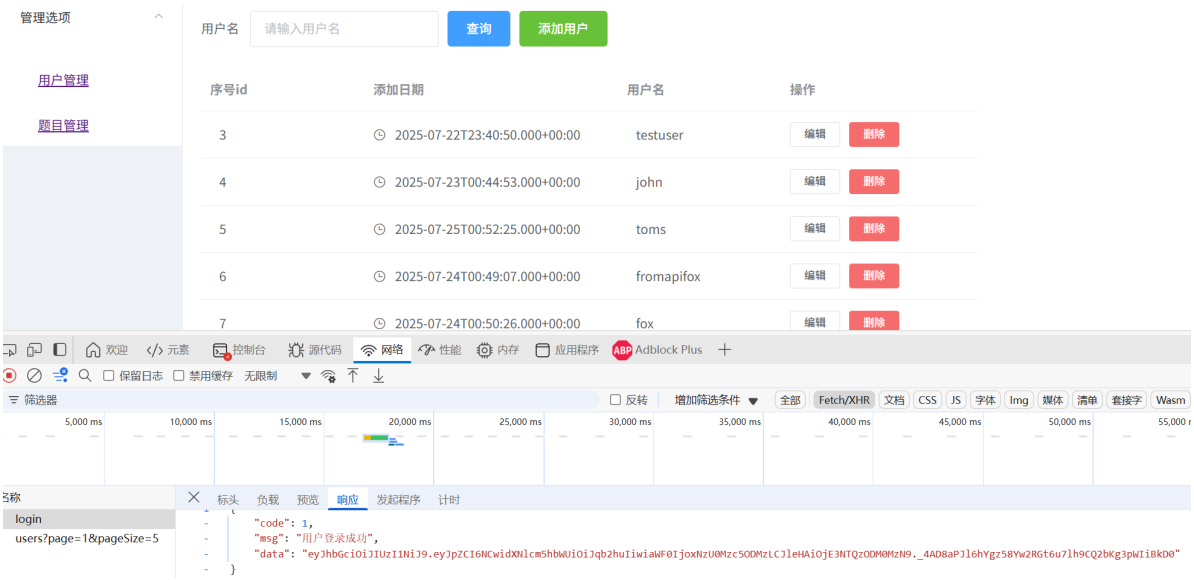
Day12-08. 登录校验-JWT令牌-登录后下发令牌哔哩哔哩bilibili

后端登录成功生成**Jwt**令牌：

Controller中的代码：

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

            String username = loginData.get("username");
            String password = loginData.get("password");

            if (StringUtils.isAnyBlank(username, password)) {
                return Result.error("用户名或密码为空");
            }
            User userResult = userService.login(username, password);
            if(userResult!=null){
                Claims claims = Jwts.claims();
                claims.put("id", userResult.getId());
                claims.put("username", userResult.getUserName());

                String token = JwtUtil.generateTokenWithClaims(claims);
                Result result = Result.success("用户登录成功");
                result.setData(token);
                return result;
            }else{
                return Result.error("用户登录失败");
            }
        }

```

前端测试后，可以获取Jwt Token:



前端存储**Jwt**令牌到本地

```
  login() {
    axios.post('login', {
      username: this.form.username,
      password: this.form.password
    }, {
      headers: {
```

```
          'Content-Type': 'application/json'
        }
      })
      .then(res => {
        console.log('登录响应:', res.data);
        if (res.data.code === 1) {
          const token = res.data.data;
          localStorage.setItem('jwt_token', token); //将jwt_token保存到本地;
          this.$router.push('/user'); // 登录成功跳转
        } else {
          this.$message.error('登录失败: ' + res.data.message);
        }
      })
      .catch(err => {
        console.error(err);
        this.$message.error('登录异常');
      });
    }
  },
```

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：

```
import axios from 'axios';

// 添加请求拦截器
axios.interceptors.request.use(
  (config) => {
    // 从 localStorage 中获取 token
    const token = localStorage.getItem('jwt_token');
    if (token) {
      // 在请求头中添加 token 字段
      config.headers.token = token;
    }
    return config;
  },
  (error) => {
    // 请求错误处理
    return Promise.reject(error);
  }
);
```

后端拦截类**Filter**：

新建filter目录，创建JwtFilter类作为Filter的实现类：

```
package com.tfzhang.quiz.filter;

import jakarta.servlet.*;
import jakarta.servlet.annotation.WebFilter;

import java.io.IOException;

@WebFilter(urlPatterns = "/*")
    public class JwtFilter implements Filter {

```

```
        //初始化，只调用一次；
        public void init(FilterConfig filterConfig) throws ServletException
{
            System.out.println("JwtFilter init");
        }

        @Override  //拦截请求；每当有新请求时；
        public void doFilter(ServletRequest req, ServletResponse res,
FilterChain chain) throws IOException, ServletException {
            System.out.println("JwtFilter拦截到请求");
            chain.doFilter(req, res);
        }

        //销毁，只调用一次；
        public void destroy() {
            System.out.println("JwtFilter destroy");
        }
    }

```

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：

```
//拦截所有类型的请求；
@WebFilter(urlPatterns = "/*")

//对请求进行放行；
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：

```
@ServletComponentScan //使得Filter发挥作用；
@SpringBootApplication
public class QuizApplication {

    public static void main(String[] args) {
        SpringApplication.run(QuizApplication.class, args);
    }

}
```

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：

```
package com.tfzhang.quiz.filter;

import com.alibaba.fastjson.JSONObject;
import com.tfzhang.quiz.model.Result;
```

```java
import jakarta.servlet.*;
import jakarta.servlet.annotation.WebFilter;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.util.StringUtils;

import java.io.IOException;

import com.tfzhang.quiz.utils.JwtUtil;

@WebFilter(urlPatterns = "/*")
public class JwtFilter implements Filter {

    public void init(FilterConfig filterConfig) throws ServletException {
        System.out.println("JwtFilter init");
    }

    @Override
    public void doFilter(ServletRequest req, ServletResponse res,
FilterChain chain) throws IOException, ServletException {
        System.out.println("JwtFilter拦截到请求");
//        chain.doFilter(req, res);

        HttpServletRequest request = (HttpServletRequest) req;
        HttpServletResponse response = (HttpServletResponse) res;

        //1、获取请求url;
        String url=request.getRequestURL().toString();
        //2、判断url中是否包含login,如果包含，则说明是登录操作，放行;
        if(url.contains("login")){
            chain.doFilter(request, response);
            return;
        }

        //3、获取请求头中的令牌(token)
        String token = request.getHeader("token");

        //4、判断令牌是否存在，如果不存在，返回未登录信息。
        if(!StringUtils.hasLength(token)){
            response.setStatus(HttpServletResponse.SC_UNAUTHORIZED); // 设置
状态码为 401
            Result result = Result.error("NOT_LOGIN");
            //手动将对象转为json，并传回前端;
            String noLogin= JSONObject.toJSONString(result);
            response.setContentType("application/json;charset=UTF-8");
            res.getWriter().write(noLogin);
            return;
        }

        //5、解析token，如果解析失败，返回未登录信息。
        try {
            JwtUtil.parseToken(token);//只要解析不成功，就说明有问题;
        }catch(Exception e){
            e.printStackTrace();
            response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
            Result result = Result.error("NOT_LOGIN");
            //手动将对象转为json，并传回前端;
            String noLogin=JSONObject.toJSONString(result);
```

```
            response.setContentType("application/json;charset=UTF-8");
            res.getWriter().write(noLogin);
            return;
        }

        //6、放行。
        chain.doFilter(req,res);
    }

    public void destroy() {
        System.out.println("JwtFilter destroy");
    }
}
```

其中的JSONObject来自于：

```
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.83</version>
        </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：

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import UserView from '../views/elment/UserView.vue'
import QuestionView from '@/views/elment/QuestionView.vue'
import LoginView from '@/views/elment/LoginView.vue'
// import { meta } from '@babel/eslint-parser'

Vue.use(VueRouter)

const routes = [
  {
    path: '/',
    redirect: '/login'
  },
  {
    path:'/user',
    name: 'user',
    component: UserView,
    meta: {
      requiresAuth: true // 需要登录才能访问
    }
  },
  {
    path:'/question',
    name:'question',
    component:QuestionView,
```

```
    meta: {
      requiresAuth: true // 需要登录才能访问
    }
  },
  {
    path:'/login',
    name:'login',
    component: LoginView
  }
]

const router = new VueRouter({
  routes
})

// 全局导航守卫
router.beforeEach((to, from, next) => {
  // 检查路由是否需要登录
  if (to.matched.some(record => record.meta.requiresAuth)) {
    const token = localStorage.getItem('jwt_token'); // 从 localStorage 获取
token
    if (!token) {
      // 如果没有 token，跳转到登录页面
      next('/login');
    } else {
      // 如果有 token，继续访问目标页面
      next();
    }
  } else {
    // 如果路由不需要登录，直接继续
    next();
  }
});
export default router
```

6、添加前端退出功能

添加按钮组件：

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

js脚本中添加logout()方法：

```
    logout() {
      // 删除 token
      localStorage.removeItem('jwt_token');
      // 跳转到登录页面
      this.$router.push('/login');
      this.$message({
        type: 'success',
        message: '已退出登录',
      });
    },
```

7、跨域访问：

要配置对应的类：

```
package com.tfzhang.quiz.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class CorsConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
            .allowedOrigins("http://localhost:8081")
            .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
            .allowedHeaders("*")
            .allowCredentials(true);
    }
}
```

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)：

```
    // proxy: {
    //   '/': {
    //     target: 'http://localhost:8080',
    //     changeOrigin: true,
    //     ws: true,
    //     pathRewrite: {
    //       '^/': ''
    //     }
    //   }
    // }
```

对于axios的设置：

```
axios.get('/login', {
  withCredentials: true
});
//或者统一配置:
axios.defaults.withCredentials = true;
```

main.js设置前端的baseUrl:

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

chatgpt的地址:

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