第16章

微人事项目实战

本章概要

- 微人事项目介绍
- 项目技术架构
- 前后端分离项目构建
- 登录模块实现
- 动态加载用户菜单
- 邮件发送
- 员工资料导入导出
- 在线聊天
- 前端项目打包

本章将通过一个前后端分离项目带读者掌握目前流行的Spring Boot+Vue前后端分离开发环境的搭建以及项目的开发流程。本章重点向读者介绍前后端分离环境的搭建以及开发流程，也涉及少量的业务逻辑。本章项目的完整代码可以在GitHub上下载，下载地址为https://github.com/lenve/vhr，本章在展示代码时仅展示项目关键步骤的核心代码。

16.1 项目简介

人事管理系统是一种常见的企业后台管理系统，它的主要目的是加强各个部门之间的协调和提高工作效率。人事管理系统提供了员工资料管理、人事管理、工资管理、统计管理以及系统管理

等功能，通过人事管理系统，人事组织部门能做到以人为中心，各部门之间实现资源共享，并且实现即时通信，提高工作效率，简化烦琐的手工统计、信息汇总和工资业务等大量的人工工作，让人事组织和工资管理工作在人事组织相关的各部门之间活跃起来。

16.2　技术架构

本项目采用当下流行的前后端分离的方式开发，后端使用 Spring Boot 开发，前端使用 Vue+ElementUI 来构建 SPA。SPA 是指 Single-Page Application，即单页面应用，SPA 应用通过动态重写当前页面来与用户交互，而非传统的从服务器重新加载整个新页面。这种方法避免了页面之间切换打断用户体验，使应用程序更像一个桌面应用程序。在 SPA 中，所有的 HTML、JavaScript 和 CSS 都通过单个页面的加载来检索，或者根据用户操作动态装载适当的资源并添加到页面。在 SPA 中，前端将通过 Ajax 与后端通信。对于开发者而言，SPA 最直观的感受就是项目开发完成后，只有一个 HTML 页面，所有页面的跳转都通过路由进行导航。前后端分离的另一个好处是一个后端可以对应多个前端，由于后端只负责提供数据，前后端的交互都是通过 JSON 数据完成的，因此后端开发成功后，前端可以是 PC 端页面，也可以是 Android、iOS 以及微信小程序等。

16.2.1　Vue 简介

> Vue (读音 /vju：/，类似于 view) 是一套用于构建用户界面的渐进式框架。与其他大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。另一方面，当与现代化的工具链以及各种支持类库结合使用时，Vue 完全能够为复杂的单页应用提供驱动。
>
> ——Vue 官网

对于 Vue 的基础知识，本书不做过多介绍，由于 Vue 的文档都是中文文档，因此强烈建议初学者通读官方文档来了解 Vue 的基本使用方法（地址为 https://cn.vuejs.org/v2/guide/），本书后面将直接介绍 Vue 在项目中的使用。

16.2.2　Element 简介

Vue 桌面端组件库非常多，比较流行的有 Element、Vux、iView、mint-ui、muse-ui 等，本项目采用 Element 作为前端页面组件库。要说设计，这些 UI 库差异都不是很大，基本上都是 Material Design 风格的，本项目采用 Element 主要考虑到该库的使用人数较多（截至写作本书时，Element 在 GitHub 上的 star 数已达 29 000，接近 30 000），出了问题容易找到解决方案。关于 Element 的用法，强烈建议初学者通读官方文档学习（地址为 http://element-cn.eleme.io/#/zh-CN/component）。

16.2.3 其他

除了前端技术点外，后端用到的技术主要就是第 1~15 章提到的技术，这里就不详细展开了。

16.3 项目构建

16.3.1 前端项目构建

Vue 项目使用 webpack 来构建。首先确保本地已经安装了 NodeJS，然后在 CMD 中执行如下命令，可以创建并启动一个名为 vuehr 的前端项目：

```
1  npm install -g vue-cli
2  vue init webpack vuehr
3  cd vuehr
4  npm run dev
```

在执行“vue init webpack vuehr”命令时，会要求依次输入项目的基本信息，如图 16-1 所示。

```
? Project name vuehr
? Project description A Vue.js project
? Author 江南一点雨 <wangsong0210@gmail.com>
? Vue build standalone
? Install vue-router? Yes
? Use ESLint to lint your code? No
? Set up unit tests No
? Setup e2e tests with Nightwatch? No
? Should we run `npm install` for you after the project has been created? (recommended) npm

   vue-cli · Generated "vuehr".
```

图 16-1

基本信息主要包括：

- 项目名称。
- 项目描述。
- 项目作者。
- Vue 项目构建：运行+编译还是仅运行。
- 是否安装 vue-router。
- 是否使用 ESLint。
- 是否使用单元测试。
- 是否适用 Nightwatch e2e 测试。
- 是否在项目创建成功后自动执行“npm install”安装依赖，若选择否，则在第 4 行命令执行之前执行“npm install”。

当“npm run dev”命令执行之后，在浏览器中输入 http://localhost:8080，显示页面如图 16-2 所示。

图 16-2

16.3.2　后端项目构建

后端使用 Spring Boot 创建一个 Spring Boot 工程，添加 spring-boot-starter-web 依赖即可：

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

当然，后端所需的依赖不止 spring-boot-starter-web，在后文功能不断完善的过程中，再继续添加其他依赖。另外，后端项目所需的 Redis 配置、邮件发送配置、POI 配置、WebSocket 配置等，将在涉及相关功能时向读者介绍。

16.3.3　数据模型设计

完整的数据库脚本可以在 GitHub 上下载，下载地址为 https://github.com/lenve/vhr/blob/master/hrserver/src/main/resources/vhr.sql，这里仅展示本项目的数据字典。

adjustsalary 表（员工调薪表）如表 16-1 所示。

表 16-1　adjustsalary 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键
eid		Integer	外键，普通索引	员工 id
asDate		Date		调薪日期
beforeSalary		Integer		调前薪资
afterSalary		Integer		调后薪资
reason		String(255)		调薪原因
remark		String(255)		备注

appraise 表（员工评价表）如表 16-2 所示。

表 16-2 appraise 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键
eid		Integer	外键，普通索引	员工 id
appDate		Date		考评日期
appResult		String(32)		考评结果
appContent		String(255)		考评内容
remark		String(255)		备注

department 表（部门表）如表 16-3 所示。

表 16-3 department 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键
name		String(32)		部门名称
parentId		Integer		父部门 id
depPath		String(255)		部门 path
enabled		Enum	默认值：1	是否可用
isParent		Enum	默认值：0	是否为父部门

employee 表（员工信息表）如表 16-4 所示。

表 16-4 employee 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	员工编号
name		String(10)		员工姓名
gender		String(4)		性别
birthday		Date		出生日期
idCard		String(18)		身份证号
wedlock		String(2)		婚姻状况
nationId		Integer(8)	外键，普通索引	民族
nativePlace		String(20)		籍贯
politicId		Integer(8)	外键，普通索引	政治面貌
email		String(20)		邮箱
phone		String(11)		电话号码
address		String(64)		联系地址
departmentId		Integer	外键，普通索引	所属部门
jobLevelId		Integer	外键，普通索引	职称 ID
posId		Integer	外键，普通索引	职位 ID
engageForm		String(8)		聘用形式
tiptopDegree		String(2)		最高学历

（续表）

字段名	逻辑名	数据类型	约束	说明
specialty		String(32)		所属专业
school		String(32)		毕业院校
beginDate		Date		入职日期
workState		String(2)	默认值：在职	在职状态
worked		String(8)	普通索引	工号
contractTerm		Float		合同期限
conversionTime		Date		转正日期
notWorkDate		Date		离职日期
beginContract		Date		合同起始日期
endContract		Date		合同终止日期
workAge		Integer		工龄

employeeec 表（员工奖惩表）如表 16-5 所示。

表 16-5　employeeec 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键
eid		Integer	外键，普通索引	员工编号
ecDate		Date		奖罚日期
ecReason		String(255)		奖罚原因
ecPoint		Integer		奖罚分
ecType		Integer		奖罚类别，0：奖，1：罚
remark		String(255)		备注

employeeremove 表（员工调岗表）如表 16-6 所示。

表 16-6　employeeremove 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键
eid		Integer	外键，普通索引	员工 id
afterDepId		Integer		调动后部门
afterJobId		Integer		调动后职位
removeDate		Date		调动日期
reason		String(255)		调动原因
remark		String(255)		备注

employeetrain 表（员工培训表）如表 16-7 所示。

表 16-7　employeetrain 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键

（续表）

字段名	逻辑名	数据类型	约束	说明
eid		Integer	外键，普通索引	员工编号
trainDate		Date		培训日期
trainContent		String(255)		培训内容
remark		String(255)		备注

empsalary 表（员工薪资关联表）如表 16-8 所示。

表 16-8 empsalary 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键
eid		Integer	外键，普通索引	员工 id
sid		Integer	外键，普通索引	薪资 id

hr 表（hr 表）如表 16-9 所示。

表 16-9 hr 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	hrID
name		String(32)		姓名
phone		String(11)		手机号码
telephone		String(16)		住宅电话
address		String(64)		联系地址
enabled		Enum	默认值：1	账户是否可用
username		String(255)		用户名
password		String(255)		密码
userface		String(255)		用户头像
remark		String(255)		备注

hr_role 表（hr 角色表）如表 16-10 所示。

表 16-10 hr_role 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键 id
hrid		Integer	外键，普通索引	操作员 id
rid		Integer	外键，普通索引	角色 id

joblevel 表（职称表）如表 16-11 所示。

表 16-11 joblevel 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键
name		String(32)		职称名称

（续表）

字段名	逻辑名	数据类型	约束	说明
titleLevel		String(3)		职称级别
createDate		Date	默认值：CURRENT_TIMESTAMP	创建日期
enabled		Enum	默认值：1	是否可用

menu 表（菜单表）如表 16-12 所示。

表 16-12　menu 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键
url		String(64)		请求路径规则
path		String(64)		路由 path
component		String(64)		组件名称
name		String(64)		组件名
iconCls		String(64)		菜单图标
keepAlive		Enum		菜单切换时是否保活
requireAuth		Enum		是否登录后才能访问
parentId		Integer	外键，普通索引	父菜单 id
enabled		Enum	默认值：1	是否可用

menu_role 表（菜单角色关联表）如表 16-13 所示。

表 16-13　menu_role 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键
mid		Integer	外键，普通索引	菜单 id
rid		Integer	外键，普通索引	角色 id

msgcontent 表（消息内容表）如表 16-14 所示。

表 16-14　msgcontent 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键
title		String(64)		消息标题
message		String(255)		消息内容
createDate		Date	非空，默认值：CURRENT_TIMESTAMP	创建日期

nation 表（民族表）如表 16-15 所示。

表 16-15 nation 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键
name		String(32)		名称

oplog 表（操作日志表）如表 16-16 所示。

表 16-16 oplog 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键
addDate		Date		添加日期
operate		String(255)		操作内容
hrid		Integer	外键，普通索引	操作员 ID

politicsstatus 表（政治面貌表）如表 16-17 所示。

表 16-17 politicsstatus 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键
name		String(32)		名称

position 表（职位表）如表 16-18 所示。

表 16-18 position 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键
Name		String(32)	唯一索引	职位
createDate		Date	默认值：CURRENT_TIMESTAMP	创建日期
enabled		Enum	默认值：1	是否可用

role 表（角色表）如表 16-19 所示。

表 16-19 role 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键
name		String(64)		角色名称
nameZh		String(64)		角色中文名称

salary 表（薪水表）如表 16-20 所示。

表 16-20 salary 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键

（续表）

字段名	逻辑名	数据类型	约束	说明
basicSalary		Integer		基本工资
bonus		Integer		奖金
lunchSalary		Integer		午餐补助
trafficSalary		Integer		交通补助
allSalary		Integer		应发工资
pensionBase		Integer		养老金基数
pensionPer		Float(12,31)		养老金比率
createDate		Date		启用时间
medicalBase		Integer		医疗基数
medicalPer		Float(12,31)		医疗保险比率
accumulationFundBase		Integer		公积金基数
accumulationFundPer		Float(12,31)		公积金比率
name		String(32)		账套名称

sysmsg 表（系统消息表）如表 16-21 所示。

表 16-21　sysmsg 表

字段名	逻辑名	数据类型	约束	说明
id		Integer	主键，自增长	主键
mid		Integer	外键，普通索引	消息 id
type		Integer	默认值：0	0 表示群发消息
hrid		Integer	外键，普通索引	这条消息是给谁的
state		Integer	默认值：0	0：未读，1：已读

经过以上准备工作，项目环境就已经基本搭建成功了。另外，对于 Redis 的安装、启动等，读者可以参考第 6 章，这里不再赘述。

16.4　登 录 模 块

16.4.1　后端接口实现

后端权限认证采用 Spring Security 实现（本小节中大量知识点与第 10 章的内容相关，需要读者熟练掌握第 10 章的内容），数据库访问使用 MyBatis，同时使用 Redis 实现认证信息缓存。因此，后端首先添加如下依赖（依次是 MyBatis 依赖、Spring Security 依赖、Redis 依赖、数据库连接池依赖、数据库驱动依赖以及缓存依赖）：

```
1  <dependency>
2      <groupId>org.mybatis.spring.boot</groupId>
3      <artifactId>mybatis-spring-boot-starter</artifactId>
```

```
    <version>1.3.2</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
    <exclusions>
        <exclusion>
            <groupId>io.lettuce</groupId>
            <artifactId>lettuce-core</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
</dependency>
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.1.10</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
```

依赖添加完成后，接下来在 application.properties 中配置数据库连接、Redis 连接以及缓存等。

```
#MySQL 配置
spring.datasource.type=com.alibaba.druid.pool.DruidDataSource
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/vhr
spring.datasource.username=root
spring.datasource.password=root
#MyBatis 日志配置
mybatis.config-location=classpath:/mybatis-config.xml
#Redis 配置
spring.redis.database=0
spring.redis.host=192.168.66.130
spring.redis.port=6379
spring.redis.password=123@456
spring.redis.jedis.pool.max-active=8
spring.redis.jedis.pool.max-idle=8
spring.redis.jedis.pool.max-wait=-1ms
```

```
spring.redis.jedis.pool.min-idle=0
#缓存配置
spring.cache.cache-names=menus_cache
spring.cache.redis.time-to-live=1800s
#端口配置
server.port=8082
```

配置完成后，接下来实现用户认证的配置。用户认证使用 Spring Security 实现，因此需要首先提供一个 UserDetails 的实例，在人事管理系统中，登录操作是 Hr 登录，根据前面的 Hr 表创建 Hr 实体类并实现 UserDetails 接口，代码如下：

```
public class Hr implements UserDetails {
    private Long id;
    private String name;
    private String phone;
    private String telephone;
    private String address;
    private boolean enabled;
    private String username;
    private String password;
    private String remark;
    private List<Role> roles;
    private String userface;
    @Override
    public boolean isEnabled() {
        return enabled;
    }
    @Override
    public String getUsername() {
        return username;
    }
    @JsonIgnore
    @Override
    public boolean isAccountNonExpired() {
        return true;
    }
    @JsonIgnore
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }
    @JsonIgnore
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }
    @JsonIgnore
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        List<GrantedAuthority> authorities = new ArrayList<>();
```

```
        for (Role role : roles) {
            authorities.add(new SimpleGrantedAuthority(role.getName()));
        }
        return authorities;
    }
    @JsonIgnore
    @Override
    public String getPassword() {
        return password;
    }
    //省略getter/setter
}
```

代码解释：

- 自定义类继承自 UserDetails，并实现该接口中相关的方法。前端用户在登录成功后，需要获取当前登录用户的信息，对于一些敏感信息不必返回，使用@JsonIgnore 注解即可。
- 对于 isAccountNonExpired、isAccountNonLocked、isCredentialsNonExpired，由于 Hr 表并未设计相关字段，因此这里直接返回 true，isEnabled 方法则根据实际情况返回。
- roles 属性中存储了当前用户的所有角色信息，在 getAuthorities 方法中，将这些角色转换为 List<GrantedAuthority>的实例返回。

接下来提供一个 UserDetailsService 实例用来查询用户，代码如下：

```
@Service
public class HrService implements UserDetailsService {
    @Autowired
    HrMapper hrMapper;
    @Override
    public UserDetails loadUserByUsername(String s) throws UsernameNotFoundException{
        Hr hr = hrMapper.loadUserByUsername(s);
        if (hr == null) {
            throw new UsernameNotFoundException("用户名不存在");
        }
        return hr;
    }
}
```

自定义 HrService 实现 UserDetailsService 接口，并实现该接口中的 loadUserByUsername 方法，loadUserByUsername 方法是根据用户名查询用户的所有信息，包括用户的角色，如果没有查到相关用户，就抛出 UsernameNotFoundException 异常，表示用户不存在，如果查到了，就直接返回，由 Spring Security 框架完成密码的比对操作。

接下来需要实现动态配置权限，因此还需要提供 FilterInvocationSecurityMetadataSource 和 AccessDecisionManager 的实例。

FilterInvocationSecurityMetadataSource 代码如下：

```
@Component
public class CustomMetadataSource implements FilterInvocationSecurityMetadataSource
{
```

```
    @Autowired
    MenuService menuService;
    AntPathMatcher antPathMatcher = new AntPathMatcher();
    @Override
    public Collection<ConfigAttribute> getAttributes(Object o) {
        String requestUrl = ((FilterInvocation) o).getRequestUrl();
        List<Menu> allMenu = menuService.getAllMenu();
        for (Menu menu : allMenu) {
            if (antPathMatcher.match(menu.getUrl(), requestUrl)
&&menu.getRoles().size()>0) {
                List<Role> roles = menu.getRoles();
                int size = roles.size();
                String[] values = new String[size];
                for (int i = 0; i < size; i++) {
                    values[i] = roles.get(i).getName();
                }
                return SecurityConfig.createList(values);
            }
        }
        return SecurityConfig.createList("ROLE_LOGIN");
    }
    @Override
    public Collection<ConfigAttribute> getAllConfigAttributes() {
        return null;
    }
    @Override
    public boolean supports(Class<?> aClass) {
        return FilterInvocation.class.isAssignableFrom(aClass);
    }
}
```

代码解释：

- 在 getAttributes 方法中首先提取出请求 URL，根据请求 URL 判断该请求需要的角色信息。
- 通过 MenuService 中的 getAllMenu 方法获取所有的菜单资源进行比对，考虑到 getAttributes 方法在每一次请求中都会调用，因此可以将 getAllMenu 方法的返回值缓存下来，下一次请求时直接从缓存中获取。
- 对于所有未匹配成功的请求，默认都是登录后访问。

AccessDecisionManager 代码如下：

```
@Component
public class UrlAccessDecisionManager implements AccessDecisionManager {
    @Override
    public void decide(Authentication auth, Object o, Collection<ConfigAttribute>
cas){
        Iterator<ConfigAttribute> iterator = cas.iterator();
        while (iterator.hasNext()) {
            ConfigAttribute ca = iterator.next();
            String needRole = ca.getAttribute();
```


```
            if ("ROLE_LOGIN".equals(needRole)) {
                if (auth instanceof AnonymousAuthenticationToken) {
                    throw new BadCredentialsException("未登录");
                } else
                    return;
            }
            Collection<? extends GrantedAuthority> authorities =
auth.getAuthorities();
            for (GrantedAuthority authority : authorities) {
                if (authority.getAuthority().equals(needRole)) {
                    return;
                }
            }
        }
        throw new AccessDeniedException("权限不足!");
    }
    @Override
    public boolean supports(ConfigAttribute configAttribute) {
        return true;
    }
    @Override
    public boolean supports(Class<?> aClass) {
        return true;
    }
}
```

代码解释：

- 在 decide 方法中判断当前用户是否具备请求需要的角色，若该方法在执行过程中未抛出异常，则说明请求可以通过；若抛出异常，则说明请求权限不足。
- 如果所需要的角色是 ROLE_LOGIN，那么只需要判断 auth 不是匿名用户的实例，即表示当前用户已登录。

接下来提供一个 AccessDeniedHandler 的实例来返回授权失败的信息：

```
@Component
public class AuthenticationAccessDeniedHandler implements AccessDeniedHandler {
    @Override
    public void handle(HttpServletRequest httpServletRequest, HttpServletResponse
resp,
                       AccessDeniedException e) throws IOException {
        resp.setStatus(HttpServletResponse.SC_FORBIDDEN);
        resp.setContentType("application/json;charset=UTF-8");
        PrintWriter out = resp.getWriter();
        RespBean error = RespBean.error("权限不足，请联系管理员!");
        out.write(new ObjectMapper().writeValueAsString(error));
        out.flush();
        out.close();
    }
}
```

当授权失败时，在这里返回授权失败的信息。

当所有准备工作完成后，接下来配置 Spring Security，代码如下：

```
@Configuration
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    HrService hrService;
    @Autowired
    CustomMetadataSource metadataSource;
    @Autowired
    UrlAccessDecisionManager urlAccessDecisionManager;
    @Autowired
    AuthenticationAccessDeniedHandler deniedHandler;
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(hrService)
                .passwordEncoder(new BCryptPasswordEncoder());
    }
    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring().antMatchers("/index.html", "/static/**", "/login_p");
    }
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
        .withObjectPostProcessor(new
ObjectPostProcessor<FilterSecurityInterceptor>() {
            @Override
            public <O extends FilterSecurityInterceptor> O postProcess(O o) {
                o.setSecurityMetadataSource(metadataSource);
                o.setAccessDecisionManager(urlAccessDecisionManager);
                return o;
            }
        })
        .and()
        .formLogin().loginPage("/login_p").loginProcessingUrl("/login")
        .usernameParameter("username").passwordParameter("password")
        .failureHandler(new AuthenticationFailureHandler() {
            @Override
            public void onAuthenticationFailure(HttpServletRequest req,
                                HttpServletResponse resp,
                                AuthenticationException e) throws IOException {
                resp.setContentType("application/json;charset=utf-8");
                RespBean respBean = null;
                if (e instanceof BadCredentialsException ||
                        e instanceof UsernameNotFoundException) {
                    respBean = RespBean.error("账户名或者密码输入错误!");
                } else if (e instanceof LockedException) {
                    respBean = RespBean.error("账户被锁定，请联系管理员!");
```


```
48                } else if (e instanceof CredentialsExpiredException) {
49                    respBean = RespBean.error("密码过期，请联系管理员!");
50                } else if (e instanceof AccountExpiredException) {
51                    respBean = RespBean.error("账户过期，请联系管理员!");
52                } else if (e instanceof DisabledException) {
53                    respBean = RespBean.error("账户被禁用，请联系管理员!");
54                } else {
55                    respBean = RespBean.error("登录失败!");
56                }
57                resp.setStatus(401);
58                ObjectMapper om = new ObjectMapper();
59                PrintWriter out = resp.getWriter();
60                out.write(om.writeValueAsString(respBean));
61                out.flush();
62                out.close();
63            }
64        })
65        .successHandler(new AuthenticationSuccessHandler() {
66            @Override
67            public void onAuthenticationSuccess(HttpServletRequest req,
68                                HttpServletResponse resp,
69                                Authentication auth) throws IOException {
70            resp.setContentType("application/json;charset=utf-8");
71            RespBean respBean = RespBean.ok("登录成功!", HrUtils.getCurrentHr());
72            ObjectMapper om = new ObjectMapper();
73            PrintWriter out = resp.getWriter();
74            out.write(om.writeValueAsString(respBean));
75            out.flush();
76            out.close();
77            }
78        })
79        .permitAll()
80        .and()
81        .logout().permitAll()
82        .and().csrf().disable()
83        .exceptionHandling().accessDeniedHandler(deniedHandler);
84    }
}
```

代码解释：

- 首先通过@EnableGlobalMethodSecurity 注解开启基于注解的安全配置，启用@PreAuthorize 和@PostAuthorize 两个注解。
- 在配置类中注入之前创建的 4 个 Bean，在 AuthenticationManagerBuilder 中配置 userDetailsService 和 passwordEncoder。
- 在 WebSecurity 中配置需要忽略的路径。
- 在 HttpSecurity 中配置拦截规则、表单登录、登录成功或失败的响应等。
- 最后通过 accessDeniedHandler 配置异常处理。

另外，前文提到 MenuService 中的 getAllMenu 方法在每次请求时都需要查询数据库，效率极低，因此可以将该数据缓存下来，代码如下：

```
@Service
@Transactional
@CacheConfig(cacheNames = "menus_cache")
public class MenuService {
    @Autowired
    MenuMapper menuMapper;
    @Cacheable(key = "#root.methodName")
    public List<Menu> getAllMenu(){
        return menuMapper.getAllMenu();
    }
}
```

注　意

这里使用方法名作为缓存的 key，另外需要在项目启动类上添加@EnableCaching 注解开启缓存。

经过前面这一整套的配置后，登录认证接口已经搭建成功了，接下来可以使用 Postman 等工具进行测试了。

登录测试如图 16-3 所示。

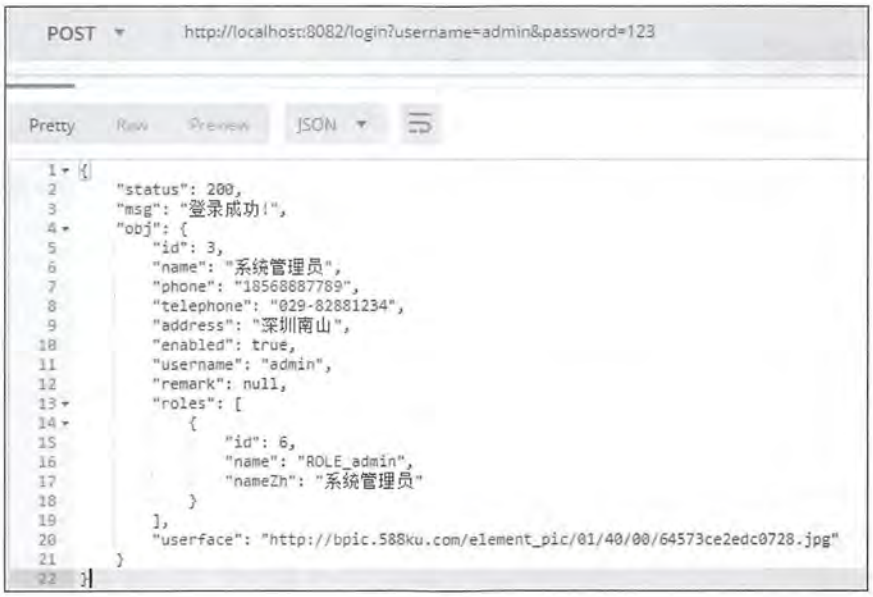

图 16-3

登录成功后，访问 http://localhost:8082/employee/advanced/hello 接口，由于当前用户不具备相应的角色，访问结果如图 16-4 所示。

图 16-4

若访问 http://localhost:8082/employee/basic/hello 地址，则可以看到正常的结果，如图 16-5 所示。

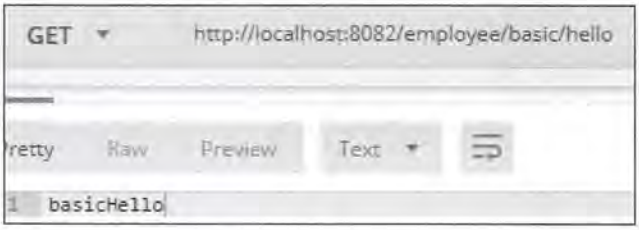

图 16-5

确认后端接口均可以正常运行后，接下来开发前端。

16.4.2 前端实现

1. 引入 Element 和 Axios

前端 UI 使用 Element，网络请求则使用 Axios，因此首先安装 Element 和 Axios 依赖，代码如下：

```
npm i element-ui -S
npm i axios -S
```

依赖添加成功后，接下来在 main.js 中引入 Element，代码如下：

```
import ElementUI from 'element-ui'
import 'element-ui/lib/theme-chalk/index.css'
Vue.use(ElementUI)
```

引入 Element 之后，接下来就可以在项目中直接使用相关组件了。

对于网络请求，由于在每一次请求时都需要判断各种异常情况，然后提示用户，例如请求是否成功、失败的原因等，考虑到这些判断基本上都使用重复的代码，因此可以将网络请求封装，做成 Vue 的插件方便使用。由于封装的代码比较长，这里就不贴出来了，读者可以在 GitHub 上查看，地址为 https://github.com/lenve/vhr/blob/master/vuehr/src/utils/api.js。配置完成后，在 main.js 中导入封装的方法，然后配置为 Vue 的 prototype，代码如下：

```
import {getRequest} from './utils/api'
import {postRequest} from './utils/api'
```

```
import {deleteRequest} from './utils/api'
import {putRequest} from './utils/api'
Vue.prototype.getRequest = getRequest;
Vue.prototype.postRequest = postRequest;
Vue.prototype.deleteRequest = deleteRequest;
Vue.prototype.putRequest = putRequest;
```

配置完成后，接下来对于任何需要使用网络请求的地址，都可以使用 this.XXX 执行一个网络请求，例如要执行登录请求，就可以通过 this.postRequest(url,param)执行。

2. 开发 Login 页面

接下来在 components 目录下创建 Login.vue 页面进行登录页面开发，代码如下：

```
<template>
<el-form :rules="rules" class="login-container" label-position="left"
         label-width="0px" v-loading="loading">
<h3 class="login_title">系统登录</h3>
<el-form-item prop="account">
<el-input type="text" v-model="loginForm.username"
              auto-complete="off" placeholder="账号"></el-input>
</el-form-item>
<el-form-item prop="checkPass">
<el-input type="password" v-model="loginForm.password"
              auto-complete="off" placeholder="密码"></el-input>
</el-form-item>
<el-checkbox class="login_remember" v-model="checked"
              label-position="left">记住密码</el-checkbox>
<el-form-item style="width: 100%">
<el-button type="primary" style="width: 100%"
@click="submitClick">登录</el-button>
</el-form-item>
</el-form>
</template>
<script>
  export default{
    data(){
      return {
        rules: {
          account: [{required: true, message: '请输入用户名', trigger: 'blur'}],
          checkPass: [{required: true, message: '请输入密码', trigger: 'blur'}]
        },
        checked: true,
        loginForm: {
          username: 'admin',
          password: '123'
        },
        loading: false
      }
    },
    methods: {
```

```
    submitClick: function () {
      var _this = this;
      this.loading = true;
      this.postRequest('/login', {
        username: this.loginForm.username,
        password: this.loginForm.password
      }).then(resp=> {
        _this.loading = false;
        if (resp && resp.status == 200) {
          var data = resp.data;
          _this.$store.commit('login', data.obj);
          var path = _this.$route.query.redirect;
          _this.$router
.replace({path: path == '/' || path == undefined ? '/home' : path});
        }
      });
    }
  }
}
</script>
<style>
  .login-container {
    border-radius: 15px;
    background-clip: padding-box;
    margin: 180px auto;
    width: 350px;
    padding: 35px 35px 15px 35px;
    background: #fff;
    border: 1px solid #eaeaea;
    box-shadow: 0 0 25px #cac6c6;
  }
  .login_title {
    margin: 0px auto 40px auto;
    text-align: center;
    color: #505458;
  }
  .login_remember {
    margin: 0px 0px 35px 0px;
    text-align: left;
  }
</style>
```

代码解释：

- 系统登录使用 Element 中的 el-form 来实现。同时使用了 Element 标签提供的校验规则。
- 当用户单击“登录”按钮时，通过 this.postRequest 方法发起一个登录请求，登录成功后，将登录的用户信息保存到 store 中，同时跳转到 Home 页，或者某个重定向页面。

3. 配置路由

登录页面开发完成后，接下来在路由中配置登录页面，代码如下：

```
import Vue from 'vue'
import Router from 'vue-router'
import Login from '@/components/Login'
import Home from '@/components/Home'
Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/',
      name: 'Login',
      component: Login,
      hidden: true
    }, {
      path: '/home',
      name: '主页',
      component: Home,
      hidden: true,
      meta: {
        requireAuth: true
      }
    }
  ]
})
```

另外，由于 main.js 是入口 JS，在 main.js 中导入了 App 组件，App 组件默认有 Vue 的 Logo，将 Logo 图片删除，只保留一个<router-view/>即可，修改后的 App.vue 如下：

```
<template>
<div id="app">
<router-view/>
</div>
</template>
```

4. 配置请求转发

最后，由于前端项目和后端项目在不同的端口下启动，前端的网络请求无法直接发送到后端，因此需要配置请求转发。下面介绍配置方式。

修改 config 目录下的 index.js 文件，修改 proxyTable，代码如下：

```
proxyTable: {
  '/': {
    target: 'http://localhost:8082',
    changeOrigin: true,
    pathRewrite: {
      '^/': ''
    }
  },
  '/ws/*': {
    target: 'ws://127.0.0.1:8082',
    ws: true
```

```
  }
},
```

这里配置了两条规则，第一条是配置 HTTP 请求转发，第二条是配置 WebSocket 请求转发，WebSocket 请求在本项目的即时通信模块中会用到。

5. 启动前端项目

做完这些操作后，接下来打开 CMD 命令窗口，进入当前项目目录下，执行如下命令启动项目：

```
npm run dev
```

如果开发者使用 WebStorm 开发前端项目，也可以单击 WebStorm 右上角的下拉按钮（见图 16-6），然后单击+，选择 npm（见图 16-7），配置 Name 和启动脚本（见图 16-8）。

图 16-6

图 16-7

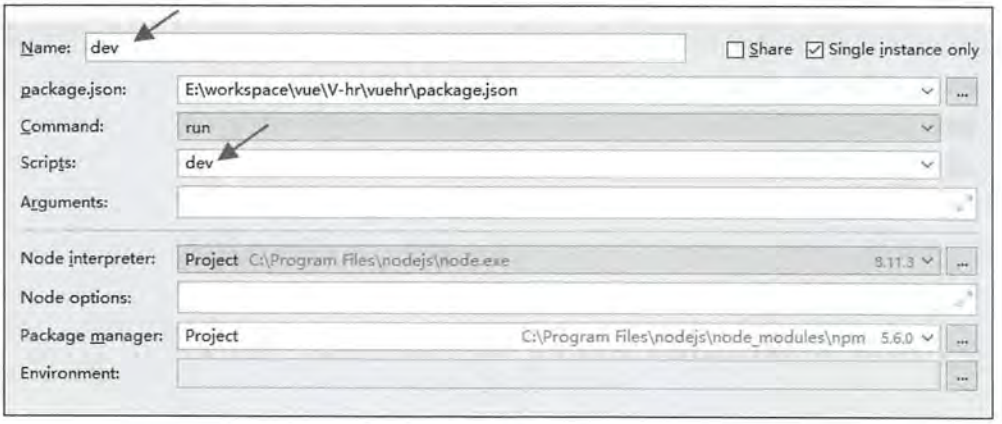

图 16-8

配置完成后，就可以直接通过单击 WebStorm 右上角的“启动”按钮启动项目了，如图 16-9 所示。

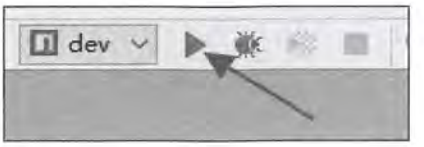

图 16-9

6. 测试

当前端项目启动成功后，接下来在浏览器中输入 http://localhost:8080，即可看到登录页面，如图 16-10 所示。

图 16-10

输入用户名和密码，单击“登录”按钮，即可登录成功，通过 Chrome 调试工具可以看到登录请求，如图 16-11 所示。

图 16-11

至此，登录功能就实现了。这里展示的只是部分核心代码，完整代码可以在 GitHub 上下载，下载地址为 https://github.com/lenve/vhr。

16.5 动态加载用户菜单

用户菜单就是用户登录成功后首页左侧显示的菜单，如图 16-12 所示。这个菜单数据是根据用户的角色动态加载的，即不同身份的用户登录成功后看到的菜单是不一样的。接下来看这个功能如何实现。

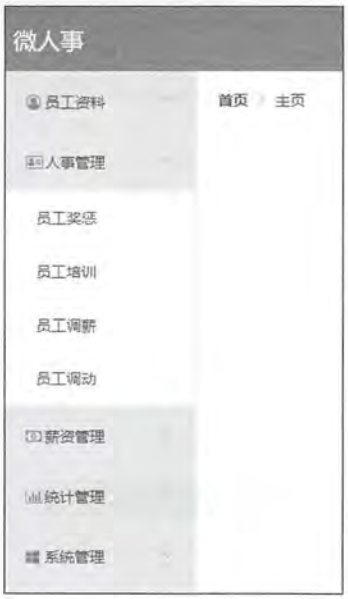

图 16-12

16.5.1 后端接口实现

后端接口的实现比较容易，根据登录用户的 id 查询该用户具有的角色，再根据角色信息查看对应的 Menu，数据模型如图 16-13 所示。

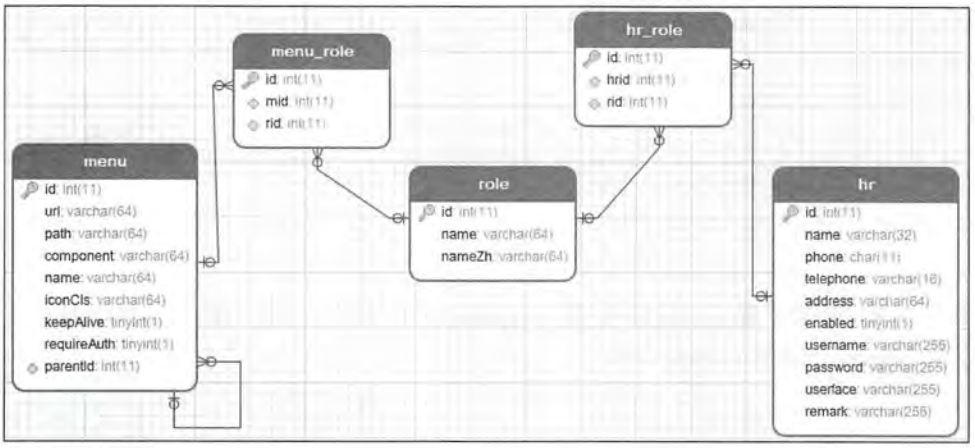

图 16-13

首先创建 MenuMapper，根据用户 id 查询 Menu，代码如下：

```
public interface MenuMapper {
    List<Menu> getMenusByHrId(Long hrId);
     //省略其他方法
}
```

对应的 MenuMapper.xml 文件中则根据当前用户 id 查询用户可以查看的角色，查询 SQL 如下（源文件过大，这里就不展示了，完整文件可以在 GitHub 上下载，下载地址为 https://github.com/lenve/vhr/blob/master/hrserver/src/main/java/org/sang/mapper/MenuMapper.xml）：

```
SELECT DISTINCT m1.*,m2.`id` AS id2,m2.`component` AS commponent2,m2.`enabled` AS
enabled2,m2.`keepAlive` AS keepAlive2,m2.`name` AS name2,m2.`path` AS path2,m2.`url`
AS url2,m2.`requireAuth` AS requireAuth2,m2.`parentId` AS parentId2 FROM menu m1,menu
m2,menu_role mr,role r,hr_role hrr  WHERE m1.`id`=m2.`parentId` AND mr.`rid`=r.`id`
AND mr.`mid`=m2.`id` AND hrr.`rid`=r.`id` AND hrr.`hrid`=#{id}
```

然后分别创建 MenuService 和 ConfigController，ConfigController 用来返回基本的系统配置信息。

MenuService 代码如下：

```
@Service
public class MenuService {
    @Autowired
    MenuMapper menuMapper;
    public List<Menu> getMenusByHrId() {
        return menuMapper.getMenusByHrId(HrUtils.getCurrentHr().getId());
    }
     //省略其他方法
}
public class HrUtils {
    public static Hr getCurrentHr() {
        return (Hr)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    }
}
```

其中，HrUtils 是一个工具方法，用来返回当前登录用户的信息。

ConfigController 代码如下：

```
@RestController
@RequestMapping("/config")
public class ConfigController {
    @Autowired
    MenuService menuService;
    @RequestMapping("/sysmenu")
    public List<Menu> sysmenu() {
        return menuService.getMenusByHrId();
    }
}
```

配置完成后，启动 Spring Boot 项目，访问 http://localhost:8082/config/sysmenu 接口，即可看到当前登录用户所能查看的菜单数据，如图 16-14 所示。

图 16-14

16.5.2 前端实现

后端返回了菜单数据，前端请求该接口获取菜单数据，这里的步骤很简单，主要分两步：

- 将服务端返回的 JSON 动态添加到当前路由中。
- 将服务端返回的 JSON 数据保存到 store 中，然后各个 Vue 页面根据 store 中的数据来渲染菜单。

这里涉及的第一个问题是请求时机，即何时去请求菜单数据。如果直接在登录成功之后请求菜单资源，那么在请求到 JSON 数据之后，将其保存在 store 中，以便下一次使用。但是这样会有一个问题：假如用户登录成功之后，单击 Home 页的某一个按钮，进入某一个子页面中，然后按一下 F5 键进行刷新，这个时候就会出现空白页面，因为按 F5 键刷新之后 store 中的数据就没了，而我们又只在登录成功的时候请求了一次菜单资源。要解决这个问题，有两种方案：方案一，不要将菜单资源保存到 store 中，而是保存到 localStorage 中，这样即使按 F5 键刷新之后数据还在；方案二，直接在每一个页面的 mounted 方法中都加载一次菜单资源。由于菜单资源是非常敏感的，因此不建议将其保存到本地，故舍弃方案一，但是方案二的工作量有点大，而且也不易维护，这里可以

使用路由中的导航守卫来简化方案二的工作量。下面介绍具体实现步骤。

首先在 store 中创建一个 routes 数组，这是一个空数组，代码如下：

```
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)

export default new Vuex.Store({
  state: {
    routes: []
  },
  mutations: {
    initMenu(state, menus){
      state.routes = menus;
    }
});
```

然后开启路由全局守卫，代码如下：

```
router.beforeEach((to, from, next)=> {
    if (to.name == 'Login') {
      next();
      return;
    }
    var name = store.state.user.name;
    if (name == '未登录') {
      if (to.meta.requireAuth || to.name == null) {
        next({path: '/', query: {redirect: to.path}})
      } else {
        next();
      }
    } else {
      initMenu(router, store);
      next();
    }
  }
)
```

代码解释：

- 这里使用 router.beforeEach 配置了一个全局前置守卫。
- 首先判断目标页面是不是 Login，若是 Login 页面，则直接通过，因为 Login 页面不需要菜单数据。
- 接下来获取 store 中保存的当前登录的用户数据，若获取到的用户名为“未登录”，则表示用户尚未登录，在用户尚未登录的情况下，如果要跳转到某一个页面，就需要判断该页面是否要求登录后才能访问，若要求了，则直接跳转到登录页面，并配置 redirect 参数。若用户已经登录，则先执行 initMenu 方法初始化菜单数据，然后通过 next();进入下一个页面。

初始化菜单的操作如下：

```
export const initMenu = (router, store)=> {
  if (store.state.routes.length > 0) {
    return;
  }
  getRequest("/config/sysmenu").then(resp=> {
    if (resp && resp.status == 200) {
      var fmtRoutes = formatRoutes(resp.data);
      router.addRoutes(fmtRoutes);
      store.commit('initMenu', fmtRoutes);
    }
  })
}
export const formatRoutes = (routes)=> {
  let fmRoutes = [];
  routes.forEach(router=> {
    let {
      path,
      component,
      name,
      meta,
      iconCls,
      children
    } = router;
    if (children && children instanceof Array) {
      children = formatRoutes(children);
    }
    let fmRouter = {
      path: path,
      component(resolve){
        if (component.startsWith("Home")) {
          require(['../components/' + component + '.vue'], resolve)
        } else if (component.startsWith("Emp")) {
          require(['../components/emp/' + component + '.vue'], resolve)
        } else if (component.startsWith("Per")) {
          require(['../components/personnel/' + component + '.vue'], resolve)
        } else if (component.startsWith("Sal")) {
          require(['../components/salary/' + component + '.vue'], resolve)
        } else if (component.startsWith("Sta")) {
          require(['../components/statistics/' + component + '.vue'], resolve)
        } else if (component.startsWith("Sys")) {
          require(['../components/system/' + component + '.vue'], resolve)
        }
      },
      name: name,
      iconCls: iconCls,
      meta: meta,
      children: children
    };
    fmRoutes.push(fmRouter);
  })
```

```
  return fmRoutes;
}
```

代码解释：

- 在初始化菜单中，首先判断 store 中的数据是否存在，如果存在，则说明这次跳转是正常的跳转，而不是用户按 F5 键或者直接在地址栏输入某个地址进入的，这时直接返回，不必执行菜单初始化。
- 若 store 中不存在菜单数据，则需要初始化菜单数据，通过 getRequest("/config/sysmenu")方法获得菜单 JSON 数据之后，首先通过 formatRoutes 方法将服务器返回的 JSON 转为 router 需要的格式，这里主要是转 component，因为服务端返回的 component 是一个字符串，而 router 中需要的却是一个组件，因此我们在 formatRoutes 方法中根据服务端返回的 component 动态加载需要的组件即可。
- 数据格式准备成功之后，一方面将数据存到 store 中，另一方面利用路由中的 addRoutes 方法将之动态添加到路由中。

加载到路由数据之后，接下来就是菜单渲染了。菜单渲染操作在 Home.vue 组件中完成，部分核心代码如下：

```
<template>
<div>
<el-container class="home-container">
<el-header class="home-header">
</el-header>
<el-container>
<el-aside width="180px" class="home-aside">
<div>
<el-menu unique-opened router>
<template v-for="(item,index) in this.routes" v-if="!item.hidden">
<el-submenu :key="index" :index="index+''">
<template slot="title">
<i :class="item.iconCls"></i>
<span slot="title">{{item.name}}</span>
</template>
<el-menu-item width="180px"
                        v-for="child in item.children"
                        :index="child.path"
                        :key="child.path">{{child.name}}
</el-menu-item>
</el-submenu>
</template>
</el-menu>
</div>
</el-aside>
<el-main>
</el-main>
</el-container>
</el-container>
```

```
</div>
</template>
<script>
  export default{
    computed: {
      user(){
        return this.$store.state.user;
      },
      routes(){
        return this.$store.state.routes
      }
    }
  }
</script>
```

代码解释：

- 在计算属性中返回 routes 数据。
- 遍历 routes 中的数据，根据 routes 中的数据渲染出 el-submenu 和 el-menu-item。

配置完成后，启动前端项目，使用不同的身份登录，登录成功后，就可以看到不同用户对应不同的操作菜单了。图 16-15 所示是系统管理员看到的菜单数据。图 16-16 所示是用户曾巩看到的菜单数据。

图 16-15

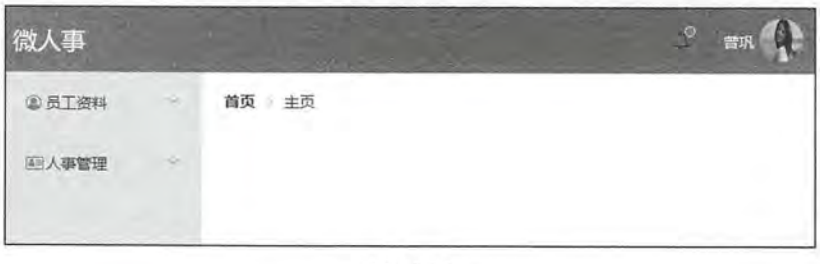

图 16-16

动态加载用户菜单就完全实现了，完整代码可以在 https://github.com/lenve/vhr 下载。

16.6　员工资料模块

完成登录模块和菜单加载模块之后，一个前后端分离的项目框架基本上就搭建成功了。接下来是业务的开发，主要是后端提供接口，前端提供页面并请求数据。下面向读者介绍员工资料模块的开发。员工资料模块页面如图 16-17 所示。

图 16-17

16.6.1　后端接口实现

员工基本资料数据的展示，后端只需要提供一个分页查询+条件查询的接口即可，代码如下：

```
@RequestMapping(value = "/emp", method = RequestMethod.GET)
public Map<String, Object> getEmployeeByPage(
      @RequestParam(defaultValue = "1") Integer page,
      @RequestParam(defaultValue = "10") Integer size,
      @RequestParam(defaultValue = "") String keywords,
      Long politicId, Long nationId, Long posId,
      Long jobLevelId, String engageForm,
      Long departmentId, String beginDateScope) {
   Map<String, Object> map = new HashMap<>();
   List<Employee> employeeByPage = empService.getEmployeeByPage(page, size,
         keywords,politicId, nationId, posId, jobLevelId, engageForm,
         departmentId, beginDateScope);
   Long count = empService.getCountByKeywords(keywords, politicId, nationId,
         posId,jobLevelId, engageForm, departmentId, beginDateScope);
   map.put("emps", employeeByPage);
   map.put("count", count);
```

```
17      return map;
18  }
```

分页查询中，page 默认为 1，size 默认为 10，查询关键字 keywords 默认为空字符串，后面几个参数则根据政治面貌、民族、职位、职称、聘用形式、部门以及入职日期查询。具体的查询代码比较简单，这里就不贴出来了。

后端接口开发成功后，先用 Postman 进行测试，结果如图 16-18 所示。

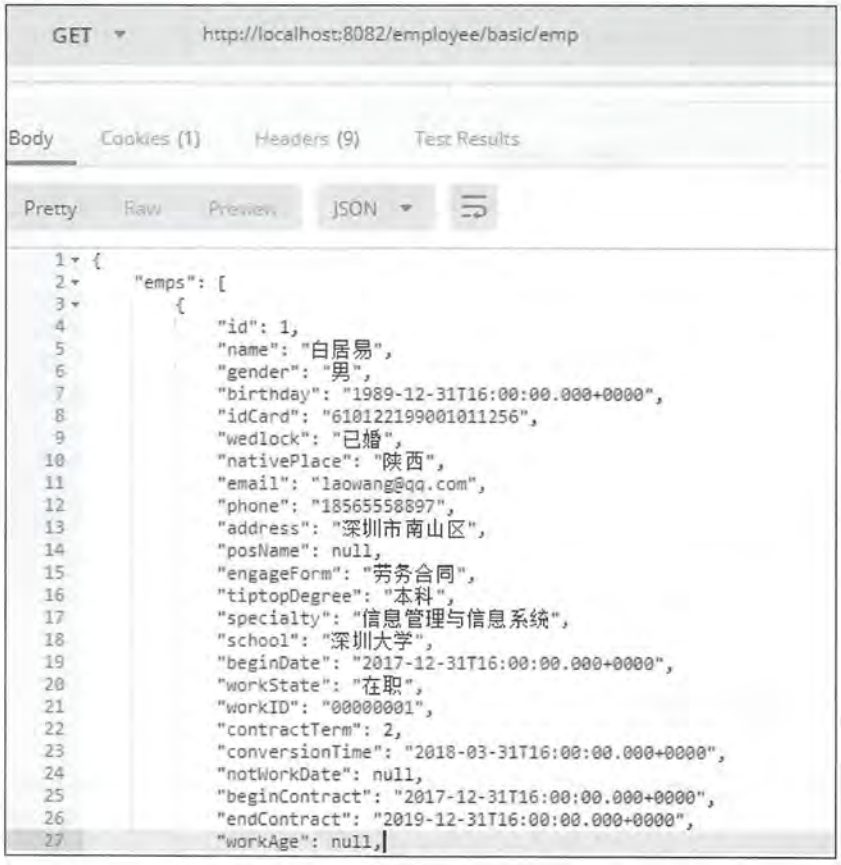

图 16-18

注 意

员工资料中的基本资料一项，接口设计规则为 “/employee/basic/**”，这是为了和数据库保持一致，防止没有权限的用户拿到请求接口后直接去请求数据。

确认后端接口可以调通后，接下来就可以开发前端页面了。

16.6.2 前端实现

前端数据的展示使用 Element 中的表格来实现。在 components 目录下创建 emp 文件夹，然后在 emp 文件夹下创建 EmpBasic.vue 用来展示前端数据。注意，这里之所以将页面放在 emp 目录下，

是为了配合 16.5 节提到的菜单数据格式化（在数据格式化时，设置员工资料相关的页面都放在 emp 目录下）。

EmpBasic.vue 核心代码如下：

```
<template>
<div>
<el-container>
<el-header>
</el-header>
<el-main>
<el-table
          :data="emps"
          v-loading="tableLoading"
          border
          stripe
          @selection-change="handleSelectionChange"
          size="mini"
          style="width: 100%">
<el-table-column
            prop="gender"
            label="性别"
            width="50">
</el-table-column>
<el-table-column
            width="85"
            align="left"
            label="出生日期">
<template slot-scope="scope">
              {{ scope.row.birthday | formatDate}}
             </template>
</el-table-column>
<el-table-column
            prop="idCard"
            width="150"
            align="left"
            label="身份证号码">
</el-table-column>

<!--省略部分代码-->


</el-table>
</div>
</el-main>
</el-container>
</div>
</template>
<script>
  export default {
    data() {
```

```
      return {
        emps: [],
        keywords: ''
        }
      };
    },
    mounted: function () {
      this.loadEmps();
    },
    methods: {
      loadEmps(){
        var _this = this;
        this.tableLoading = true;
        this.getRequest("/employee/basic/emp?page=" + this.currentPage
          + "&size=10&keywords=" + this.keywords).then(resp=> {
          this.tableLoading = false;
          if (resp && resp.status == 200) {
            var data = resp.data;
            _this.emps = data.emps;
            _this.totalCount = data.count;
          }
        })
      }
    }
  };
</script>
```

代码解释：

- 首先在 data 中定义 emps 用来存放所有员工的 JSON 数据，然后定义 keywords 用来存放查询的关键字。
- 在加载该页面时，在 mounted 中调用 loadEmps 初始化员工数据。

另外注意，表格中的日期使用 formatDate 过滤器实现日期的格式化，formatDate 是一个全局过滤器。

formatDate 过滤器定义如下：

```
import Vue from 'vue'
Vue.filter("formatDate", formatDate);
function formatDate(value) {
  var date = new Date(value);
  var year = date.getFullYear();
  var month = date.getMonth() + 1;
  var day = date.getDate();
  if (month < 10) {
    month = "0" + month;
  }
  if (day < 10) {
    day = "0" + day;
  }
```

```
    return year + "-" + month + "-" + day;
}
```

经过以上几步配置后，读者就可以看到如图 16-17 所示的效果图了。

16.7　配置邮件发送

在员工资料模块，hr 表可以手动录入员工数据，当一个员工数据被成功录入系统后，系统会自动向该员工发送一封欢迎入职邮件，要实现这个功能非常容易（关于详细的邮件发送配置，读者可以参考 13.1 节），下面介绍其具体配置。

首先在后端项目中引入邮件发送相关依赖：

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

这里除了邮件发送依赖外，还引入了 Thymeleaf 依赖，Thymeleaf 的作用是构建邮件模板。

依赖添加成功后，接下来在 application.properties 中配置邮件：

```
spring.mail.host=smtp.qq.com
spring.mail.port=465
spring.mail.username=xxx@qq.com
spring.mail.password=13.1.1 小节申请到的授权码
spring.mail.default-encoding=UTF-8
spring.mail.properties.mail.smtp.socketFactory.class=javax.net.ssl.SSLSocketFactory
spring.mail.properties.mail.debug=true
```

配置完成后，接下来在 resources/templates 目录下创建邮件模板 email.html，代码如下：

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Title</title>
</head>
<body>
<p>你好，
<span th:text="${name}"></span>同学，欢迎加入 XXX 大家庭！您的入职信息如下：</p>
<table border="1" cellspacing="0">
<tr>
<td><strong style="color: #F00">工号</strong></td>
<td th:text="${workID}"></td>
```

```
</tr>
<tr>
<td><strong style="color: #F00">合同期限</strong></td>
<td th:text="${contractTerm}+'年'"></td>
</tr>
<tr>
<td><strong style="color: #F00">合同起始日期</strong></td>
<td th:text="${#dates.format(beginContract, 'yyyy-MM-dd')}"></td>
</tr>
<tr>
<td><strong style="color: #F00">合同截至日期</strong></td>
<td th:text="${#dates.format(endContract, 'yyyy-MM-dd')}"></td>
</tr>
<tr>
<td><strong style="color: #F00">所属部门</strong></td>
<td th:text="${departmentName}"></td>
</tr>
<tr>
<td><strong style="color: #F00">职位</strong></td>
<td th:text="${posName}"></td>
</tr>
</table>
<p>
<strong style="color: #F00; font-size: 24px;">
        希望在未来的日子里，携手共进！
</strong>
</p>
</body>
</html>

```

考虑到邮件发送是一个耗时操作，因此在子线程中完成邮件发送操作，代码如下：

```
public class EmailRunnable implements Runnable {
    private Employee employee;
    private JavaMailSender javaMailSender;
    private TemplateEngine templateEngine;

    public EmailRunnable(Employee employee,
                         JavaMailSender javaMailSender,
                         TemplateEngine templateEngine) {
        this.employee = employee;
        this.javaMailSender = javaMailSender;
        this.templateEngine = templateEngine;
    }
    @Override
    public void run() {
        try {
            MimeMessage message = javaMailSender.createMimeMessage();
            MimeMessageHelper helper = new MimeMessageHelper(message, true);
            helper.setTo(employee.getEmail());
```

```
            helper.setFrom("1510161612@qq.com");
            helper.setSubject("XXX 集团：通知");
            Context ctx = new Context();
            ctx.setVariable("name", employee.getName());
            ctx.setVariable("workID", employee.getWorkID());
            ctx.setVariable("contractTerm", employee.getContractTerm());
            ctx.setVariable("beginContract", employee.getBeginContract());
            ctx.setVariable("endContract", employee.getEndContract());
            ctx.setVariable("departmentName", employee.getDepartmentName());
            ctx.setVariable("posName", employee.getPosName());
            String mail = templateEngine.process("email.html", ctx);
            helper.setText(mail, true);
            javaMailSender.send(message);
        } catch (MessagingException e) {
            System.out.println("发送失败");
        } catch (javax.mail.MessagingException e) {
            e.printStackTrace();
        }
    }
}
```

最后，在用户添加成功后，启动该线程发送邮件，代码如下：

```
@RequestMapping(value = "/emp", method = RequestMethod.POST)
public RespBean addEmp(Employee employee) {
    if (empService.addEmp(employee) == 1) {
        List<Position> allPos = positionService.getAllPos();
        for (Position allPo : allPos) {
            if (allPo.getId() == employee.getPosId()) {
                employee.setPosName(allPo.getName());
            }
        }
        executorService.execute(new EmailRunnable(employee,
                javaMailSender, templateEngine));
        return RespBean.ok("添加成功!");
    }
    return RespBean.error("添加失败!");
}
```

配置完成后，重启后端项目，此时在前端添加用户，添加完成后，系统会根据所添加用户的邮箱自动发送一封欢迎入职邮件，如图 16-19 所示。

图 16-19

完整的邮件发送代码可以在 GitHub 上查看，地址为 https://github.com/lenve/vhr。

16.8 员工资料导出

将员工资料导出为 Excel 是一个非常常见的需求，后端提供导出接口，前端下载导出数据即可。

16.8.1 后端接口实现

后端实现主要是将查询到的员工数据集合转为可以下载的 ResponseEntity<byte[]>，代码如下：

```
public static ResponseEntity<byte[]> exportEmp2Excel(List<Employee> emps) {
    HttpHeaders headers = null;
    ByteArrayOutputStream baos = null;
    try {
        //1.创建 Excel 文档
        HSSFWorkbook workbook = new HSSFWorkbook();
        //2.创建文档摘要
        workbook.createInformationProperties();
        //3.获取文档信息，并配置
        DocumentSummaryInformation dsi =
 workbook.getDocumentSummaryInformation();
        //3.1 文档类别
        dsi.setCategory("员工信息");
        //3.2 设置文档管理员
        dsi.setManager("江南一点雨");
        //3.3 设置组织机构
        dsi.setCompany("XXX 集团");
        //4.获取摘要信息并配置
        SummaryInformation si = workbook.getSummaryInformation();
        //4.1 设置文档主题
```

```
        si.setSubject("员工信息表");
        //4.2.设置文档标题
        si.setTitle("员工信息");
        //4.3 设置文档作者
        si.setAuthor("XXX 集团");
        //4.4 设置文档备注
        si.setComments("备注信息暂无");
        //创建 Excel 表单
        HSSFSheet sheet = workbook.createSheet("XXX 集团员工信息表");
        //创建日期显示格式
        HSSFCellStyle dateCellStyle = workbook.createCellStyle();
        dateCellStyle.setDataFormat(HSSFDataFormat.getBuiltinFormat("m/d/yy"));
        //创建标题的显示样式
        HSSFCellStyle headerStyle = workbook.createCellStyle();
        headerStyle.setFillForegroundColor(IndexedColors.YELLOW.index);
        headerStyle.setFillPattern(FillPatternType.SOLID_FOREGROUND);
        //定义列的宽度
        sheet.setColumnWidth(0, 5 * 256);
        sheet.setColumnWidth(1, 12 * 256);
        sheet.setColumnWidth(18, 20 * 256);
        sheet.setColumnWidth(19, 12 * 256);
        sheet.setColumnWidth(20, 8 * 256);
        sheet.setColumnWidth(21, 25 * 256);
        sheet.setColumnWidth(22, 14 * 256);
        sheet.setColumnWidth(23, 12 * 256);
        sheet.setColumnWidth(24, 12 * 256);
        //5.设置表头
        HSSFRow headerRow = sheet.createRow(0);
        HSSFCell cell0 = headerRow.createCell(0);
        cell0.setCellValue("编号");
        cell0.setCellStyle(headerStyle);
        HSSFCell cell1 = headerRow.createCell(1);
        cell1.setCellValue("姓名");
        cell1.setCellStyle(headerStyle);
        HSSFCell cell18 = headerRow.createCell(18);
        cell18.setCellValue("毕业院校");
        cell18.setCellStyle(headerStyle);
        HSSFCell cell19 = headerRow.createCell(19);
        cell19.setCellValue("入职日期");
        cell19.setCellStyle(headerStyle);
        HSSFCell cell20 = headerRow.createCell(20);
        cell20.setCellValue("在职状态");
        cell20.setCellStyle(headerStyle);
        HSSFCell cell21 = headerRow.createCell(21);
        cell21.setCellValue("邮箱");
        cell21.setCellStyle(headerStyle);
        HSSFCell cell22 = headerRow.createCell(22);
        cell22.setCellValue("合同期限(年)");
        cell22.setCellStyle(headerStyle);
        HSSFCell cell23 = headerRow.createCell(23);
```

```
71          cell23.setCellValue("合同起始日期");
72          cell23.setCellStyle(headerStyle);
73          HSSFCell cell24 = headerRow.createCell(24);
74          cell24.setCellValue("合同终止日期");
75          cell24.setCellStyle(headerStyle);
76          //6.装数据
77          for (int i = 0; i < emps.size(); i++) {
78              HSSFRow row = sheet.createRow(i + 1);
79              Employee emp = emps.get(i);
80              row.createCell(0).setCellValue(emp.getId());
81              row.createCell(1).setCellValue(emp.getName());
82              row.createCell(18).setCellValue(emp.getSchool());
83              HSSFCell beginDateCell = row.createCell(19);
84              beginDateCell.setCellValue(emp.getBeginDate());
85              beginDateCell.setCellStyle(dateCellStyle);
86              row.createCell(20).setCellValue(emp.getWorkState());
87              row.createCell(21).setCellValue(emp.getEmail());
88              row.createCell(22).setCellValue(emp.getContractTerm());
89              HSSFCell beginContractCell = row.createCell(23);
90              beginContractCell.setCellValue(emp.getBeginContract());
91              beginContractCell.setCellStyle(dateCellStyle);
92              HSSFCell endContractCell = row.createCell(24);
93              endContractCell.setCellValue(emp.getEndContract());
94              endContractCell.setCellStyle(dateCellStyle);
95          }
96          headers = new HttpHeaders();
97          headers.setContentDispositionFormData("attachment",
98                  new String("员工表.xls".getBytes("UTF-8"), "iso-8859-1"));
99          headers.setContentType(MediaType.APPLICATION_OCTET_STREAM);
100         baos = new ByteArrayOutputStream();
101         workbook.write(baos);
102     } catch (IOException e) {
103         e.printStackTrace();
104     }
105     return new ResponseEntity<byte[]>(baos.toByteArray(), headers,
106 HttpStatus.CREATED);
    }
```

代码解释：

- 首先构建一个 HSSFWorkbook 进行 Excel 基本信息配置，如文档信息、摘要信息等。
- 第 37~75 行配置列的宽度并设置表头。由于配置方式重复，因此这里省略了第 2~17 列的配置，完整配置可在 GitHub 上下载。
- 第 77~94 行表示遍历 emps 集合，将数据填充到 Excel 中。
- 第 97、98 行表示设置下载请求的文件名、编码等信息。

配置完成后，在下载请求接口中调用该方法即可，代码如下：

```
1   @RequestMapping(value = "/exportEmp", method = RequestMethod.GET)
2   public ResponseEntity<byte[]> exportEmp() {
```

```
    return PoiUtils.exportEmp2Excel(empService.getAllEmployees());
}
```

16.8.2　前端实现

前端的实现比较简单，当用户单击“导出”按钮时，执行如下代码发起请求，下载文件：

```
window.open("/employee/basic/exportEmp", "_parent");
```

单击按钮时，会自动弹出文件保存窗口，将文件保存即可。下载后的 Excel 如图 16-20 所示。

编号	姓名	工号	性别	出生日期	身份证号码	婚姻状况	民族	籍贯	政治面貌	电话号码	联系地址	所属部门	职称	职位
1	白晨晨	00000001	男	1990/1/1	610122199001011256	已婚	汉族	陕西	普通公民	18565558897	深圳市南山区	总办	教授	技术总监
2	陈静	00000002	女	1989/2/1	421288198902011234	已婚	汉族	海南	中共党员	18795556693	海南省海口市美兰区	财务部	助教	运营总监
3	赵琳浩	00000003	男	1993/3/4	610122199303041456	未婚	汉族	陕西	共青团员	16698887795	陕西省西安市莲湖区	技术部	助教	研发工程师
4	鹿存亮	00000004	男	1990/1/3	610122199001031456	已婚	汉族	陕西	共青团员	15612347795	陕西省西安市莲湖区	运维部	助教	运维工程师
5	姚森	00000005	男	1991/2/5	610122199102058952	已婚	汉族	河南	共青团员	14785559936	河南洛阳人民大道58号	运维部	中级工程师	运维工程师
6	云晨	00000006	女	1993/1/5	610122199301054789	已婚	汉族	陕西西安	中共党员	15644442252	陕西西安新城区	运维部	高级工程师	运维工程师
7	贾旭明	00000007	男	1993/11/11	610122199311111234	已婚	汉族	广东广州	民革党员	15644441234	广东省广州市天河区汽车	市场部	中级工程师	研发工程师
8	张新明	00000008	男	1991/2/1	610144199102014569	已婚	汉族	广东	民建会员	18979994478	广东珠海	技术部	中级工程师	研发工程师
9	薛蕾	00000009	男	1992/7/1	610144199207017895	已婚	汉族	陕西西安	普通公民	15648887741	西安市雁塔区	运维部	初级工程师	运维工程师
10	张洁	00000010	女	1990/10/9	420177199010093652	未婚	汉族	海南	民盟盟员	13695557742	海口市秀英区	运维部	高级工程师	运维工程师
11	白晨晨2	00000011	男	1990/1/1	610122199001011256	已婚	汉族	陕西	普通公民	18565558897	深圳市南山区	技术部	教授	技术总监
12	陈静2	00000012	女	1989/2/1	421288198902011234	已婚	汉族	海南	中共党员	18795556693	海南省海口市美兰区	华南市场部	助教	运营总监
13	赵琳浩2	00000013	男	1993/3/4	610122199303041456	未婚	汉族	陕西	共青团员	16698887795	陕西省西安市莲湖区	技术部	助教	研发工程师
14	鹿存亮2	00000014	男	1990/1/3	610122199001031456	已婚	汉族	陕西	共青团员	15612347795	陕西省西安市莲湖区	运维部	助教	运维工程师
15	姚森2	00000015	男	1991/2/5	610122199102058952	已婚	汉族	河南	共青团员	14785559936	河南洛阳人民大道58号	运维部	中级工程师	运维工程师
16	云晨2	00000016	女	1993/1/5	610122199301054789	已婚	汉族	陕西西安	中共党员	15644442252	陕西西安新城区	运维部	高级工程师	运维工程师
17	贾旭明2	00000017	男	1993/11/11	610122199311111234	已婚	汉族	广东广州	民革党员	15644441234	广东省广州市天河区汽车	市场部	中级工程师	研发工程师
18	王一博	00000018	男	1991/2/1	610144199102014569	已婚	汉族	广东	民建会员	18979994478	广东珠海	技术部	中级工程师	研发工程师
19	薛蕾2	00000019	男	1992/7/1	610144199207017895	已婚	汉族	陕西西安	普通公民	15648887741	西安市雁塔区	运维部	初级工程师	运维工程师
20	张洁2	00000020	女	1990/10/9	420177199010093652	未婚	汉族	海南	民盟盟员	13695557742	海口市秀英区	运维部	高级工程师	运维工程师
21	白晨晨3	00000021	男	1990/1/1	610122199001011256	已婚	汉族	陕西	普通公民	18565558897	深圳市南山区	财务部	教授	技术总监
22	陈静3	00000022	女	1989/2/1	421288198902011234	已婚	汉族	海南	中共党员	18795556693	海南省海口市美兰区	华南市场部	助教	运营总监
23	鹿存亮3	00000024	男	1990/1/3	610122199001031456	已婚	汉族	陕西	共青团员	15612347795	陕西省西安市莲湖区	运维部	助教	运维工程师
24	姚森3	00000025	男	1991/2/5	610122199102058952	已婚	汉族	河南	共青团员	14785559936	河南洛阳人民大道58号	运维部	中级工程师	运维工程师

图 16-20

经过如上配置后，员工数据导出功能就实现了，完整代码读者可以参考 https://github.com/lenve/vhr。

16.9　员工资料导入

既然有员工资料导出需求，当然也就有导入需求。对前端而言，员工资料导入就是文件上传，对后端而言，则是获取上传的文件进行解析，并把解析出来的数据保存到数据库中。

16.9.1　后端接口实现

后端主要是获取前端上传文件的流，然后进行解析，代码如下：

```
public static List<Employee> importEmp2List(MultipartFile file,
                                             List<Nation> allNations,
                                             List<PoliticsStatus> allPolitics,
                                             List<Department> allDeps,
                                             List<Position> allPos,
                                             List<JobLevel> allJobLevels) {
    List<Employee> emps = new ArrayList<>();
    try {
```

```
    HSSFWorkbook workbook =
            new HSSFWorkbook(new POIFSFileSystem(file.getInputStream()));
    int numberOfSheets = workbook.getNumberOfSheets();
    for (int i = 0; i < numberOfSheets; i++) {
    HSSFSheet sheet = workbook.getSheetAt(i);
    int physicalNumberOfRows = sheet.getPhysicalNumberOfRows();
    Employee employee;
    for (int j = 0; j < physicalNumberOfRows; j++) {
    if (j == 0) {
       continue;//标题行
    }
    HSSFRow row = sheet.getRow(j);
    if (row == null) {
       continue;//没数据
    }
    int physicalNumberOfCells = row.getPhysicalNumberOfCells();
    employee = new Employee();
    for (int k = 0; k < physicalNumberOfCells; k++) {
    HSSFCell cell = row.getCell(k);
    switch (cell.getCellTypeEnum()) {
    case STRING: {
       String cellValue = cell.getStringCellValue();
       if (cellValue == null) {
          cellValue = "";
       }
       switch (k) {
          case 1:
             employee.setName(cellValue);
             break;
          case 2:
             employee.setWorkID(cellValue);
             break;
          case 3:
             employee.setGender(cellValue);
             break;
          case 5:
             employee.setIdCard(cellValue);
             break;
          case 6:
             employee.setWedlock(cellValue);
             break;
          case 7:
             int nationIndex = allNations.indexOf(new Nation(cellValue));
             employee.setNationId(allNations.get(nationIndex).getId());
             break;
          case 8:
             employee.setNativePlace(cellValue);
             break;
          case 9:
             int psIndex = allPolitics.indexOf(new PoliticsStatus(cellValue));
```


```
59                  employee.setPoliticId(allPolitics.get(psIndex).getId());
60                  break;
61              case 10:
62                  employee.setPhone(cellValue);
63                  break;
64              case 11:
65                  employee.setAddress(cellValue);
66                  break;
67              case 12:
68                  int depIndex = allDeps.indexOf(new Department(cellValue));
69                  employee.setDepartmentId(allDeps.get(depIndex).getId());
70                  break;
71              case 13:
72                  int jlIndex = allJobLevels.indexOf(new JobLevel(cellValue));
73                  employee.setJobLevelId(allJobLevels.get(jlIndex).getId());
74                  break;
75              case 14:
76                  int posIndex = allPos.indexOf(new Position(cellValue));
77                  employee.setPosId(allPos.get(posIndex).getId());
78                  break;
79              case 15:
80                  employee.setEngageForm(cellValue);
81                  break;
82              case 16:
83                  employee.setTiptopDegree(cellValue);
84                  break;
85              case 17:
86                  employee.setSpecialty(cellValue);
87                  break;
88              case 18:
89                  employee.setSchool(cellValue);
90                  break;
91              case 19:
92              case 20:
93                  employee.setWorkState(cellValue);
94                  break;
95              case 21:
96                  employee.setEmail(cellValue);
97                  break;
98          }
99      }
100     break;
101     default: {
102         switch (k) {
103             case 4:
104                 employee.setBirthday(cell.getDateCellValue());
105                 break;
106             case 19:
107                 employee.setBeginDate(cell.getDateCellValue());
108                 break;
```

```
109                 case 22:
110                     employee.setContractTerm(cell.getNumericCellValue());
111                     break;
112                 case 23:
113                     employee.setBeginContract(cell.getDateCellValue());
114                     break;
115                 case 24:
116                     employee.setEndContract(cell.getDateCellValue());
117                     break;
118             }
119         }
120             break;
121         }
122         }
123             emps.add(employee);
124         }
125         }
126         } catch (IOException e) {
127             e.printStackTrace();
128         }
129         return emps;
130     }
```

代码解释：

- 首先根据上传文件的流获取一个 HSSFWorkbook 对象，然后获取 workbook 中表单的个数，进行遍历。
- 对于每一个表单，首先获取行数，然后进行遍历，第一行是标题行，跳过，如果该行没有数据也跳过，如果该行数据正常，就获取该行的单元格个数进行遍历。
- 本案例中，单元格的格式主要分为三种，即日期、数字以及普通文本，因此在不同的 switch 分支中进行处理。
- 最后将遍历得到的员工数据集合返回。

在数据导入接口中调用 importEmp2List 方法，获取员工数据集合后，插入数据库中即可，代码如下：

```
1   @RequestMapping(value = "/importEmp", method = RequestMethod.POST)
2   public RespBean importEmp(MultipartFile file) {
3       List<Employee> emps = PoiUtils.importEmp2List(file,
4               empService.getAllNations(), empService.getAllPolitics(),
5               departmentService.getAllDeps(), positionService.getAllPos(),
6               jobLevelService.getAllJobLevels());
7       if (empService.addEmps(emps) == emps.size()) {
8           return RespBean.ok("导入成功!");
9       }
10      return RespBean.error("导入失败!");
11  }
```

16.9.2　前端实现

前端主要是一个 Excel 表格的上传，这里直接采用 Element 的文件上传控件，代码如下：

```
<el-upload
  :show-file-list="false"
  accept="application/vnd.ms-excel"
  action="/employee/basic/importEmp"
  :on-success="fileUploadSuccess"
  :on-error="fileUploadError"
  :disabled="fileUploadBtnText=='正在导入'"
  :before-upload="beforeFileUpload"
  style="display: inline">
<el-button size="mini" type="success"
  :loading="fileUploadBtnText=='正在导入'">
<i class="fa fa-lg fa-level-up"></i>
  {{fileUploadBtnText}}
</el-button>
</el-upload>
```

代码解释：

- accept 表示接收的上传文件类型。
- action 表示上传接口。
- :on-success 表示上传成功时的回调。
- :on-error 表示上传失败时的回调。
- :disabled 表示当 fileUploadBtnText 属性的值为“正在导入”时禁用上传控件。这个配置主要考虑到上传是一个耗时操作，在一个文件上传的过程中，其他文件暂时不能上传。
- :before-upload 表示文件上传前的回调。
- el-button 中的:loading="fileUploadBtnText=='正在导入'"表示当 fileUploadBtnText 的文本为“正在导入”时，显示一个 Loading 加载。

相关回调方法如下：

```
fileUploadSuccess(response, file, fileList){
  if (response) {
    this.$message({type: response.status, message: response.msg});
  }
  this.loadEmps();
  this.fileUploadBtnText = '导入数据';
},
fileUploadError(err, file, fileList){
  this.$message({type: 'error', message: "导入失败!"});
  this.fileUploadBtnText = '导入数据';
},
beforeFileUpload(file){
  this.fileUploadBtnText = '正在导入';
}
```

代码解释：

- 在文件上传之前，首先设置 fileUploadBtnText 的文本为“正在导入”，这样上传按钮上的文本就会变为“正在导入”，同时上传按钮的状态变为禁用，并且在上传按钮上多了一个 Loading。
- 在上传成功时，给用户以提示，然后重新加载员工数据，并将 fileUploadBtnText 的文本设置为“导入数据”。
- 上传出错时，给用户以提示，同时将 fileUploadBtnText 的文本设置为“导入数据”。

所有配置完成后，单击“导入数据”，选择 16.8 节导出的用户数据进行导入，如图 16-21 所示。

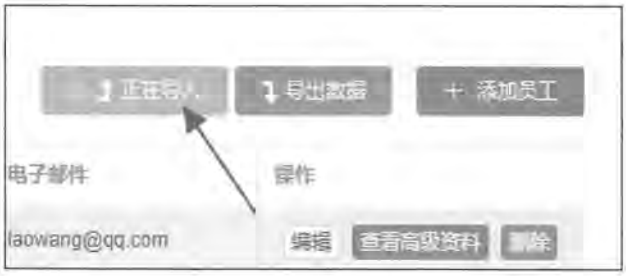

图 16-21

16.10 在线聊天

在线聊天是一个为了方便 HR 进行快速沟通提高工作效率而开发的功能，考虑到一个公司中的 HR 并不多，并发量不大，因此这里直接使用最基本的 WebSocket 来完成该功能。

16.10.1 后端接口实现

要使用 WebSocket，首先引入 WebSocket 依赖：

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

依赖添加成功后，接下来配置 WebSocket 配置类，代码如下：

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig extends AbstractWebSocketMessageBrokerConfigurer {
    @Override
    public void registerStompEndpoints(StompEndpointRegistry stompEndpointRegistry)
{
        stompEndpointRegistry.addEndpoint("/ws/endpointChat").withSockJS();
    }
```

```
9
10      @Override
11      public void configureMessageBroker(MessageBrokerRegistry registry) {
12          registry.enableSimpleBroker("/queue","/topic");
13      }
    }
```

然后创建消息转发 Controller，代码如下：

```
1   @Configuration
2   @EnableWebSocketMessageBroker
3   public class WebSocketConfig extends AbstractWebSocketMessageBrokerConfigurer {
4       @Override
5       public void registerStompEndpoints(StompEndpointRegistry stompEndpointRegistry)
6   {
7           stompEndpointRegistry.addEndpoint("/ws/endpointChat").withSockJS();
8       }
9
10      @Override
11      public void configureMessageBroker(MessageBrokerRegistry registry) {
12          registry.enableSimpleBroker("/queue","/topic");
13      }
    }
```

配置完成后，重启后端项目，然后开始配置前端。

16.10.2　前端实现

聊天功能写在 FriendChat.vue 组件中，但是用户登录成功后，首先加载的是 Home.vue 页面，在 Home 页面的右上角有一个通知的图标，如果有最新的通知，这里会显示一个红点，如图 16-22 所示。

图 16-22

因此，虽然聊天是在 FriendChat.vue 页面进行的，但是 WebSocket 连接却需要登录成功后才建立，这里选择在 store 中建立 WebSocket 请求，代码如下：

```
1   import Vue from 'vue'
2   import Vuex from 'vuex'
3   import '../lib/sockjs'
4   import '../lib/stomp'
5
6   Vue.use(Vuex)
```


```
7
8   export default new Vuex.Store({
9     state: {
10      routes: [],
11      msgList: [],
12      isDotMap: new Map(),
13      currentFriend: {},
14      stomp: Stomp.over(new SockJS("/ws/endpointChat")),
15      nfDot: false
16    },
17    mutations: {
18      toggleNFDot(state, newValue){
19        state.nfDot = newValue;
20      },
21      updateMsgList(state, newMsgList){
22        state.msgList = newMsgList;
23      },
24      updateCurrentFriend(state, newFriend){
25        state.currentFriend = newFriend;
26      },
27      addValue2DotMap(state, key){
28        state.isDotMap.set(key, "您有未读消息")
29      },
30      removeValueDotMap(state, key){
31        state.isDotMap.delete(key);
32      }
33    },
34    actions: {
35      connect(context){
36        context.state.stomp = Stomp.over(new SockJS("/ws/endpointChat"));
37        context.state.stomp.connect({}, frame=> {
38          context.state.stomp.subscribe("/user/queue/chat", message=> {
39            //接收在线聊天消息
40          });
41          context.state.stomp.subscribe("/topic/nf", message=> {
42            //接收系统通知
43          });
44        }, failedMsg=> {
45
46        });
47      }
48    }
49  });
```

定义好之后，在初始化菜单数据的地方调用 connect 方法建立 WebSocket 连接，代码如下：

```
1   export const initMenu = (router, store)=> {
2     if (store.state.routes.length > 0) {
3       return;
4     }
5     getRequest("/config/sysmenu").then(resp=> {
```

```
    if (resp && resp.status == 200) {
      var fmtRoutes = formatRoutes(resp.data);
      router.addRoutes(fmtRoutes);
      store.commit('initMenu', fmtRoutes);
      store.dispatch('connect');
    }
  })
}
```

通过 store.dispatch('connect');调用 connect 方法。

这里配置完成后，重新登录，在 Chrome 控制台可以看到 WebSocket 连接已经成功建立起来了，如图 16-23 所示。

```
Opening Web Socket...                                    stomp.js?195a:145
Web Socket Opened...                                     stomp.js?195a:145
>>> CONNECT                                             stomp.js?195a:145
accept-version:1.1,1.0
heart-beat:10000,10000


<<< CONNECTED                                            stomp.js?195a:145
version:1.1
heart-beat:0,0
user-name:admin


connected to server undefined                            stomp.js?195a:145
>>> SUBSCRIBE                                           stomp.js?195a:145
id:sub-0
destination:/user/queue/chat


>>> SUBSCRIBE                                           stomp.js?195a:145
id:sub-1
destination:/topic/nf
```

图 16-23

最后，在 FriendChat.vue 中通过如下方式发送一条消息：

```
this.$store.state.stomp.send("/ws/chat", {}, this.msg + ";" +
this.currentFriend.username);
```

另外，浏览器在收到消息之后，是将消息保存在 store 中的，这样一旦收到消息，FriendChat 页面的聊天数据就会自动更新，并且，当有新消息到达时，即使用户不在 FriendChat 页面，也能及时收到通知（主页右上角的通知图标会显示小红点）。

聊天效果如图 16-24、图 16-25 所示。

图 16-24

图 16-25

这里由于前端页面代码量庞大，因此只贴出部分关键步骤的代码，完整代码读者可以在 https://github.com/lenve/vhr 下载。

这里有两个订阅，“/user/queue/chat”是用来做在线聊天的，“/topic/nf”则是为了接收系统通知。

16.11 前端项目打包

当前端项目开发完成后，执行如下命令对项目进行打包：

```
1 npm run build
```

执行结果如图 16-26 所示。

图 16-26

打包完成后，在当前工作目录下生成一个 dist 文件夹，如图 16-27 所示。将里边的 index.html

和 static 目录复制到 Spring Boot 项目的 static 目录下，如图 16-28 所示。

图 16-27

图 16-28

接下来，启动后端项目，直接在浏览器中输入 http://localhost:8082/index.html 就可以看到登录页面，如图 16-29 所示。此时就可以将该 Spring Boot 项目直接打包发布（参见第 15 章）。

图 16-29

16.12　小　　结

本章向读者介绍了一个微人事项目，主要从登录模块、动态加载用户菜单、员工资料模块、邮件发送模块、Excel 导入导出模块、在线聊天模块以及编译打包几个方面介绍。由于原项目代码量庞大，本章主要选取一些关键步骤进行介绍，完整代码读者可以在 GitHub 上下载，下载地址为 https://github.com/lenve/vhr。