# Springboot+Vue 前后端分离-----仓库管理系统

**【从零撸一个前后端分离项目】**

明世隐技术交流QQ群：161964698

## 项目概述

后端：Springboot  mybatis-plus  java

前端：nodejs 、 vue脚手架、element-ui

数据库：mysql

up主辛苦录制，大家帮忙三连，非常的感谢大家!

## 一、创建后端项目

1. 创建文件夹
2. 打开这个文件夹(idea)
3. 创建新模块（springboot）
4. 创建测试类并测试

## 二、 加入mybatis-plus支持

1. 加入依赖代码

```
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.4.1</version>
</dependency>
```

2. 创建数据库实例、创建用户表、插入默认数据

```
create table user
(
    id       int auto_increment comment '主键'
        primary key,
    no       varchar(20)            null comment '账号',
    name     varchar(100)           not null comment '名字',
    password varchar(20)            not null comment '密码',
    age      int                    null,
    sex      int                    null comment '性别',
    phone    varchar(20)            null comment '电话',
    role_id  int                    null comment '角色 0超级管理员，1管理员，2普通账号',
```

```
    isValid  varchar(4) default 'Y' null comment '是否有效，Y有效，其他无效'
)
    charset = utf8;
```

3. yml文件的配置

端口和数据源的配置

```
server:
  port: 8090

spring:
  datasource:
    url: jdbc:mysql://localhost:3306/wms?useUnicode=true&characterEncoding=utf-
8&useSSL=false&serverTimezone=GMT%2B8
    driver-class-name: com.mysql.jdbc.Driver
    username: root
    password: root
```

4. 编写测试代码

- 创建实体类

```
@Data
public class User {
    private int id;
    private String no;
    private String name;
    private String password;
    private int sex;
    private int roleId;
    private String phone;
    private String isvalid;
}
```

- 创建mapper接口

```
@Mapper
public interface UserMapper extends BaseMapper<User> {
    public List<User> selectAll();
}
```

- 创建service接口

```
public interface UserService extends IService<User> {
    public List<User> selectAll();
}
```

- 创建service实现类

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

    @Resource
    private UserMapper userMapper;
    @Override
    public List<User> selectAll() {
        return userMapper.selectAll();
    }
}
```

- 创建配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.wms.mapper.UserMapper">
<select id="selectAll" resultType="com.wms.entity.User">
    select * from  user
</select>
</mapper>
```

- 测试类

```
@RestController
public class HelloController {
    @Autowired
    private UserService userService;

    @GetMapping
    public List<User> hello(){
        return userService.selectAll();
    }
}
```

# 三、使用代码生成器生成代码

1. 加入依赖

```
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-generator</artifactId>
    <version>3.4.1</version>
</dependency>
<dependency>
    <groupId>org.freemarker</groupId>
    <artifactId>freemarker</artifactId>
    <version>2.3.30</version>
</dependency>
<dependency>
    <groupId>com.spring4all</groupId>
    <artifactId>spring-boot-starter-swagger</artifactId>
    <version>1.5.1.RELEASE</version>
</dependency>
```

2. 获取生成器代码

```
package com.wms.common;

import com.baomidou.mybatisplus.core.exceptions.MybatisPlusException;
import com.baomidou.mybatisplus.core.toolkit.StringPool;
import com.baomidou.mybatisplus.core.toolkit.StringUtils;
import com.baomidou.mybatisplus.generator.AutoGenerator;
import com.baomidou.mybatisplus.generator.InjectionConfig;
import com.baomidou.mybatisplus.generator.config.*;
import com.baomidou.mybatisplus.generator.config.po.TableInfo;
import com.baomidou.mybatisplus.generator.config.rules.NamingStrategy;
import com.baomidou.mybatisplus.generator.engine.FreemarkerTemplateEngine;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class CodeGenerator {
    /**
     * <p>
     * 读取控制台内容
     * </p>
     */
    public static String scanner(String tip) {
        Scanner scanner = new Scanner(System.in);
        StringBuilder help = new StringBuilder();
        help.append("请输入" + tip + "：");
        System.out.println(help.toString());
        if (scanner.hasNext()) {
            String ipt = scanner.next();
```

```
            if (StringUtils.isNotBlank(ipt)) {
                return ipt;
            }
        }
        throw new MybatisPlusException("请输入正确的" + tip + "! ");
    }

    /**
     * 操作步骤:
     *  1.修改数据源包括地址密码信息，对应代码标记：一、 下同
     *  2.模块配置，可以修改包名
     *  3.修改模板（这步可忽略）
     * @param args
     */
    public static void main(String[] args) {
        // 代码生成器
        AutoGenerator mpg = new AutoGenerator();

        // 全局配置
        GlobalConfig gc = new GlobalConfig();
        String projectPath = System.getProperty("user.dir")+"/wms";
        gc.setOutputDir(projectPath + "/src/main/java");
        gc.setAuthor("wms");
        gc.setOpen(false);
        gc.setSwagger2(true); //实体属性 Swagger2 注解
        gc.setBaseResultMap(true);// XML ResultMap
        gc.setBaseColumnList(true);// XML columList
        //去掉service接口首字母的I，如DO为User则叫UserService
        gc.setServiceName("%sService");
        mpg.setGlobalConfig(gc);

        // 数据源配置
        DataSourceConfig dsc = new DataSourceConfig();
        // 一、修改数据源
        dsc.setUrl("jdbc:mysql://localhost:3306/wms01?
useUnicode=true&characterEncoding=UTF8&useSSL=false");
        // dsc.setSchemaName("public");
        dsc.setDriverName("com.mysql.jdbc.Driver");
        dsc.setUsername("root");
        dsc.setPassword("root");
        mpg.setDataSource(dsc);

        // 包配置
        PackageConfig pc = new PackageConfig();
        //pc.setModuleName(scanner("模块名"));
        // 二、模块配置
        pc.setParent("com.wms")
            .setEntity("entity")
            .setMapper("mapper")
```

```
            .setService("service")
            .setServiceImpl("service.impl")
            .setController("controller");
        mpg.setPackageInfo(pc);

        // 自定义配置
        InjectionConfig cfg = new InjectionConfig() {
            @Override
            public void initMap() {
                // to do nothing
            }
        };

        // 如果模板引擎是 freemarker
        String templatePath = "templates/mapper.xml.ftl";
        // 如果模板引擎是 velocity
        // String templatePath = "/templates/mapper.xml.vm";

        // 自定义输出配置
        List<FileOutConfig> focList = new ArrayList<>();
        // 自定义配置会被优先输出
        focList.add(new FileOutConfig(templatePath) {
            @Override
            public String outputFile(TableInfo tableInfo) {
                // 自定义输出文件名 ， 如果你 Entity 设置了前后缀、此处注意 xml 的名称会跟着发生变
化！！
                return projectPath + "/src/main/resources/mapper/" + pc.getModuleName()
                        + "/" + tableInfo.getEntityName() + "Mapper" +
StringPool.DOT_XML;
            }
        });
        /*
        cfg.setFileCreate(new IFileCreate() {
            @Override
            public boolean isCreate(ConfigBuilder configBuilder, FileType fileType,
String filePath) {
                // 判断自定义文件夹是否需要创建
                checkDir("调用默认方法创建的目录，自定义目录用");
                if (fileType == FileType.MAPPER) {
                    // 已经生成 mapper 文件判断存在，不想重新生成返回 false
                    return !new File(filePath).exists();
                }
                // 允许生成模板文件
                return true;
            }
        });
        */
        cfg.setFileOutConfigList(focList);
        mpg.setCfg(cfg);
```

```
        // 配置模板
        TemplateConfig templateConfig = new TemplateConfig();

        // 配置自定义输出模板
        //指定自定义模板路径，注意不要带上.ftl/.vm, 会根据使用的模板引擎自动识别
        // 三、修改模板
        /*templateConfig.setEntity("templates/entity2.java");
        templateConfig.setService("templates/service2.java");
        templateConfig.setController("templates/controller2.java");
        templateConfig.setMapper("templates/mapper2.java");
        templateConfig.setServiceImpl("templates/serviceimpl2.java");*/

        templateConfig.setXml(null);
        mpg.setTemplate(templateConfig);

        // 策略配置
        StrategyConfig strategy = new StrategyConfig();
        strategy.setNaming(NamingStrategy.underline_to_camel);
        strategy.setColumnNaming(NamingStrategy.underline_to_camel);
        // strategy.setSuperEntityClass("你自己的父类实体,没有就不用设置!");
        //strategy.setSuperEntityClass("BaseEntity");
        strategy.setEntityLombokModel(true);
        strategy.setRestControllerStyle(true);
        // 公共父类
        //strategy.setSuperControllerClass("BaseController");
        // strategy.setSuperControllerClass("你自己的父类控制器,没有就不用设置!");
        // 写于父类中的公共字段
        // strategy.setSuperEntityColumns("id");
        strategy.setInclude(scanner("表名，多个英文逗号分割").split(","));
        strategy.setControllerMappingHyphenStyle(true);
        //strategy.setTablePrefix(pc.getModuleName() + "_");
        // 忽略表前缀tb_,比如说tb_user,直接映射成user对象
        // 四、注意是否要去掉表前缀
        //strategy.setTablePrefix("tb_");
        mpg.setStrategy(strategy);
        mpg.setTemplateEngine(new FreemarkerTemplateEngine());
        mpg.execute();
    }
}
```

3. 生成代码并测试

# 四、实现增删改查

# 五、分页的处理

- 入参的封装

  > 看自己喜好

- 添加分页拦截器

```
@Configuration
public class MybatisPlusConfig {
    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor() {
        MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();
        interceptor.addInnerInterceptor(new PaginationInnerInterceptor(DbType.MYSQL));
        return interceptor;
    }
}
```

- 编写分页的mapper 方法

  > 加入IPage作为参数

- 自定义SQL使用Wrapper

  > 查官网

# 六、返给前端数据的封装

> 让前端收到统一的数据，方便处理

{

Code:200 //400

Msg:"成功、失败",

Total:10

Data:[] {}

}

# 七、创建前端项目

> 需要安装 node

# 八、前端Vue项目导入 IDEA并运行

# 九、导入Element-ui

> 补充：Vue脚手架(注意版本冲突) npm install -g @vue/cli

1. 安装命令

> npm i element-ui -S

2. mainjs全局引入

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

3. 测试是否引入成功

# 十、搭建页面布局

> Container 布局容器

# 十一、页面布局的拆分

> 分解的好处

# 十二、编写Header头页面

1. dropdown下拉
2. 菜单伸缩图标
3. 欢迎字样
4. 去除背景，加入下边框

# 十三、菜单导航页面编写

> 一级菜单

# 十四、菜单导航页面伸缩

> 伸缩的思路

header点击图标---提交--->父组件 --改变-->aside子组件（collapse）

# 十五、安装axios与处理跨域

安装axios

> npm install axios --save

在main.js全局引入axios

> import axios from 'axios';
> Vue.prototype.$axios =axios;

跨域

```
package com.wms.common;

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("/**")
                //是否发送Cookie
                .allowCredentials(true)
                //放行哪些原始域
                .allowedOriginPatterns("*")
                .allowedMethods(new String[]{"GET", "POST", "PUT", "DELETE"})
                .allowedHeaders("*")
                .exposedHeaders("*");
    }
}
```

get使用

```
this.$axios.get('http://localhost:8090/list').then(res=>{
    console.log(res)
})
```

post使用

```
this.$axios.post('http://localhost:8091/user/listP',{}).then(res=>{
    console.log(res)
})
```

将地址设置为全局

request.js

# 十六、列表展示

| id | 账号 | 姓名 | 角色 | 性别 | 电话 | 操作 |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | sa | 超级管理员 | 超级管理员 | 男 | | 编辑 删除 |
| 2 | ming | 小明2222 | 超级管理员 | 男 | | 编辑 删除 |
| 3 | ming | 小明 | 用户 | 男 | | 编辑 删除 |
| 4 | ming | 小明 | 管理员 | 男 | 333 | 编辑 删除 |
| 5 | ming | 小明2222 | 超级管理员 | 女 | 4444 | 编辑 删除 |

1. 列表数据
2. 用tag转换列
3. header-cell-style设置表头样式
4. 加上边框
5. 按钮（编辑、删除）
6. 后端返回结果封装（Result）

# 十七、分页处理

1. 页面加上分页代码
2. 修改查询方法和参数
3. 处理翻页、设置条数逻辑（注意一个问题）

# 十八、查询处理

1. 查询的布局（包含 查询、重置按钮）
2. 输入框
3. 下拉框
4. 回车事件（查询） @keyup.enter.native
5. 重置处理

# 十九、新增

1. 新增按钮
2. 弹出窗口
3. 编写表单
4. 提交数据（提示信息、列表刷新）
5. 数据的检查

```
age: [
    {required: true, message: '请输入年龄', trigger: 'blur'},
    {min: 1, max: 3, message: '长度在 1 到 3 个位', trigger: 'blur'},
    {pattern: /^([1-9][0-9]*){1,3}$/,message: '年龄必须为正整数字',trigger: "blur"},
    {validator:checkAge,trigger: 'blur'}
],
phone: [
    {required: true,message: "手机号不能为空",trigger: "blur"},
    {pattern: /^1[3|4|5|6|7|8|9][0-9]\d{8}$/, message: "请输入正确的手机号码", trigger:
"blur"}
]
```

```
let checkAge = (rule, value, callback) => {
    if(value>150){
        callback(new Error('年龄输入过大'));
    }else{
        callback();
    }
};
```

6. 账号的唯一验证

```
let checkDuplicate =(rule,value,callback)=>{
    if(this.form.id){
        return callback();
    }
    this.$axios.get("http://localhost:8091/user/findByNo?no="+this.form.no).then(res=>{
        if(res.code==200){
            callback()
        }else{
            callback(new Error('账号已经存在'));
        }
    })
};
```

7. 表单重置

# 二十、修改

1. 传递数据到表单
2. 提交数据到后台
3. 表单重置

# 二十一、删除

1. 获取数据（id）
2. 删除确认
3. 提交到后台

# 二十二、登录

1. 登录页面
2. 后台查询代码
3. 登录页面的路由

- 安装路由插件(npm i vue-router@3.5.4 )
- 创建路由文件

```
import VueRouter from 'vue-router';

const routes =[
    {
        path:'/',
        name:'login',
        component:()=>import('../components/Login')
    }
]

const router = new VueRouter({
    mode:'history',
    routes
})

export  default router;
```

- mainjs注册

4. 主页的路由

# 二十三、退出登录

1. 展示名字
2. 退出登录事件
3. 退出跳转、清空相关数据
4. 退出确认

# 二十四、首页个人中心

1. 编写页面
2. 路由跳转
3. 路由错误解决

```
const VueRouterPush = VueRouter.prototype.push
VueRouter.prototype.push = function push (to) {
    return VueRouterPush.call(this, to).catch(err => err)
}
```

# 二十五、菜单跳转

1. 菜单增加router、高亮
2. 配置子菜单
3. 模拟动态menu

| menuClick | menuRight | menuComponent | menuIcon |
| --- | --- | --- | --- |
| adminManage | 0 | admin/AdminManage.vue | el-icon-s-custom |
| userManage | 0,1 | user/UserManage.vue | el-icon-user-solid |

# 二十六、动态路由

1. 设计menu表和数据
2. 生成menu对应的后端代码
3. 返回数据
4. vuex状态管理
    - 安装(npm i vuex@3.0.0)
    - 编写store

    ```
    import vue from 'vue'
    import Vuex from 'vuex'
    vue.use(Vuex)
    ```

    - mainjs注册

    ```
    import store from "./store"
    ```

5. 生成menu数据
6. 生成路由数据

- 获取路由列表

```
路由列表 router.options.routes
```

- 组装路由

```
routes.forEach((routeItem)=>{
    if(routeItem.path=='/Index'){
        menuList.forEach(menu=>{
            let newRoute = {
                path:'/'+menu.menuclick,
                name:menu.menuname,
                meta:{
                    title:menu.menuname
                },
                component:()=>import('../components/'+menu.menucomponent)
            }

            routeItem.children.push(newRoute);
        })
    }
})
```

- 合并路由

```
router.addRoutes(routes)
```

- 错误处理

```
export function resetRouter() {
    router.matcher = new VueRouter({
        mode:'history',
        routes: []
    }).matcher
}
```

# 二十七、管理员管理、用户管理

# 二十八、仓库管理

1. 表设计
2. 根据表生成后端代码
3. 编写后端增删改成代码
4. postman测试查询代码
5. 编写前端相关代码

# 二十九、物品类型管理

1. 表设计
2. 根据表生成后端代码

3. 编写后端增删改成代码
4. postman测试查询代码
5. 编写前端相关代码

# 三十、物品管理

1. 表设计
2. 根据表生成后端代码
3. 编写后端增删改成代码
4. postman测试查询代码
5. 编写前端相关代码

```
count: [
    {required: true, message: '请输入数量', trigger: 'blur'},
    {pattern: /^([1-9][0-9]*){1,4}$/,message: '数量必须为正整数字',trigger: "blur"},
    {validator:checkCount,trigger: 'blur'}
],
```

```
let checkCount = (rule, value, callback) => {
    if(value>9999){
        callback(new Error('数量输入过大'));
    }else{
        callback();
    }
};
```

6. 仓库和分类的列表展现
7. 查询条件中增加仓库和分类的条件
8. 表单中仓库和分类下拉实现

# 三十一、记录管理

1. 表设计
2. 根据表生成后端代码
3. 编写后端查询代码
4. 编写前端相关代码
5. 优化

- 列表展示商品名、仓库、分类名
- 按物品名查询、仓库、分类查询

# 三十二、入库、出库

1. 表单编写
2. 入库操作（记录、更新物品数量、自动填充时间）

3. 用户选择

# 三十三、优化

1. 出入库权限控制
2. 记录查询权限控制