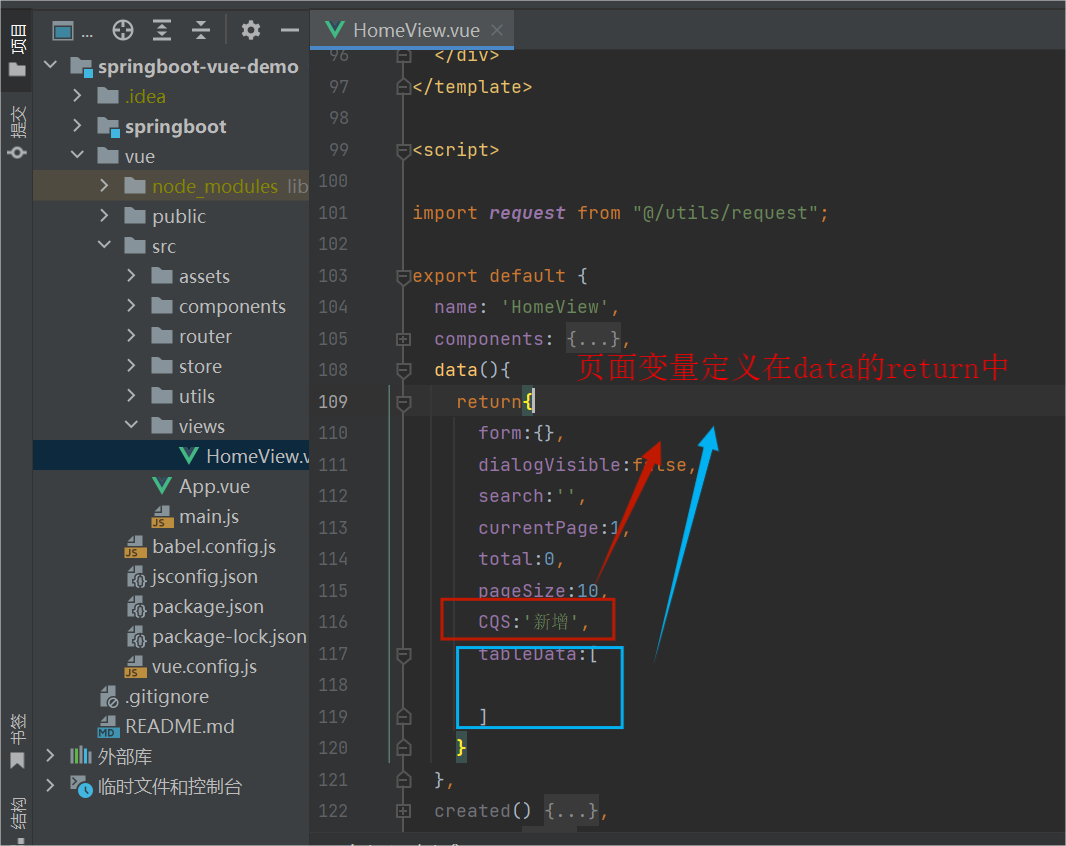
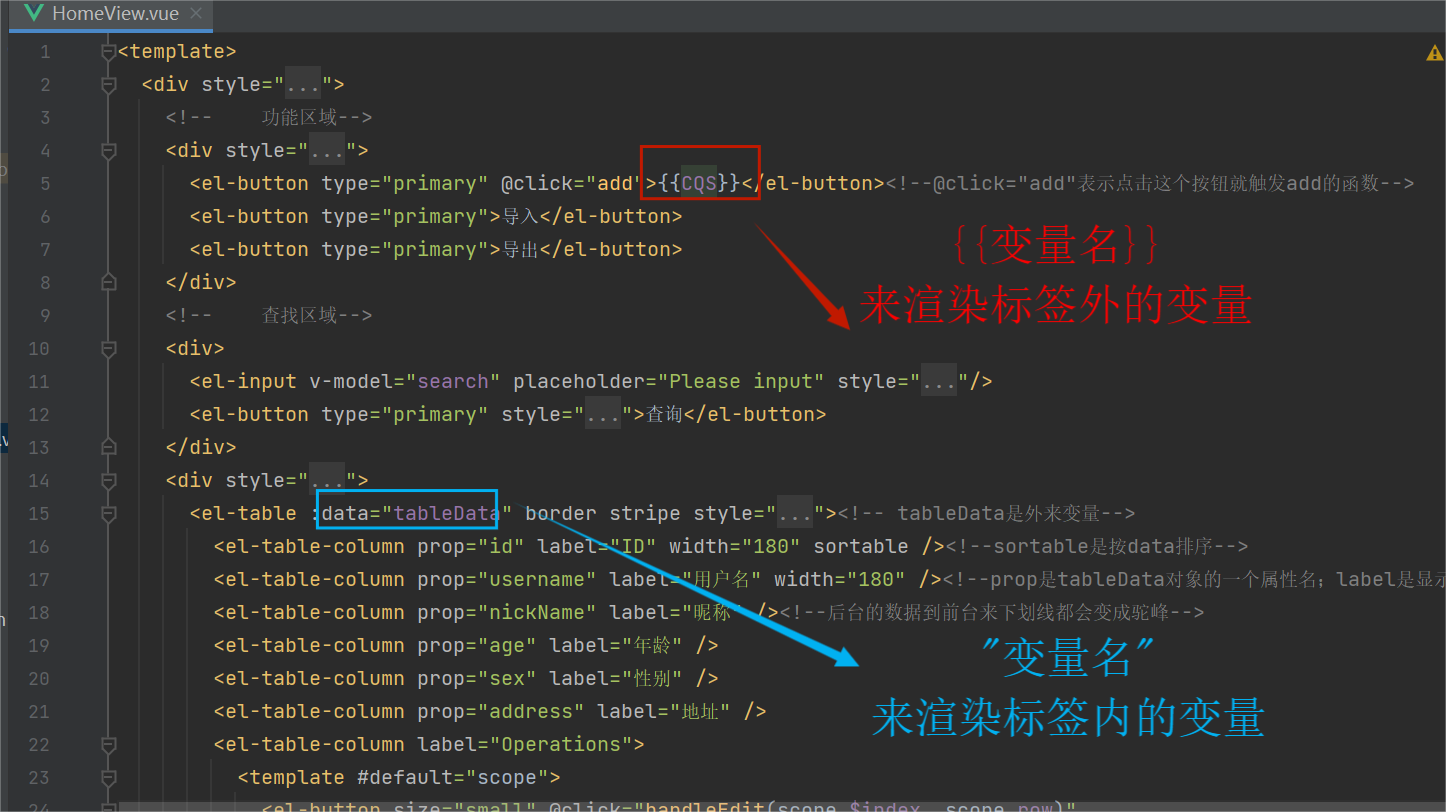
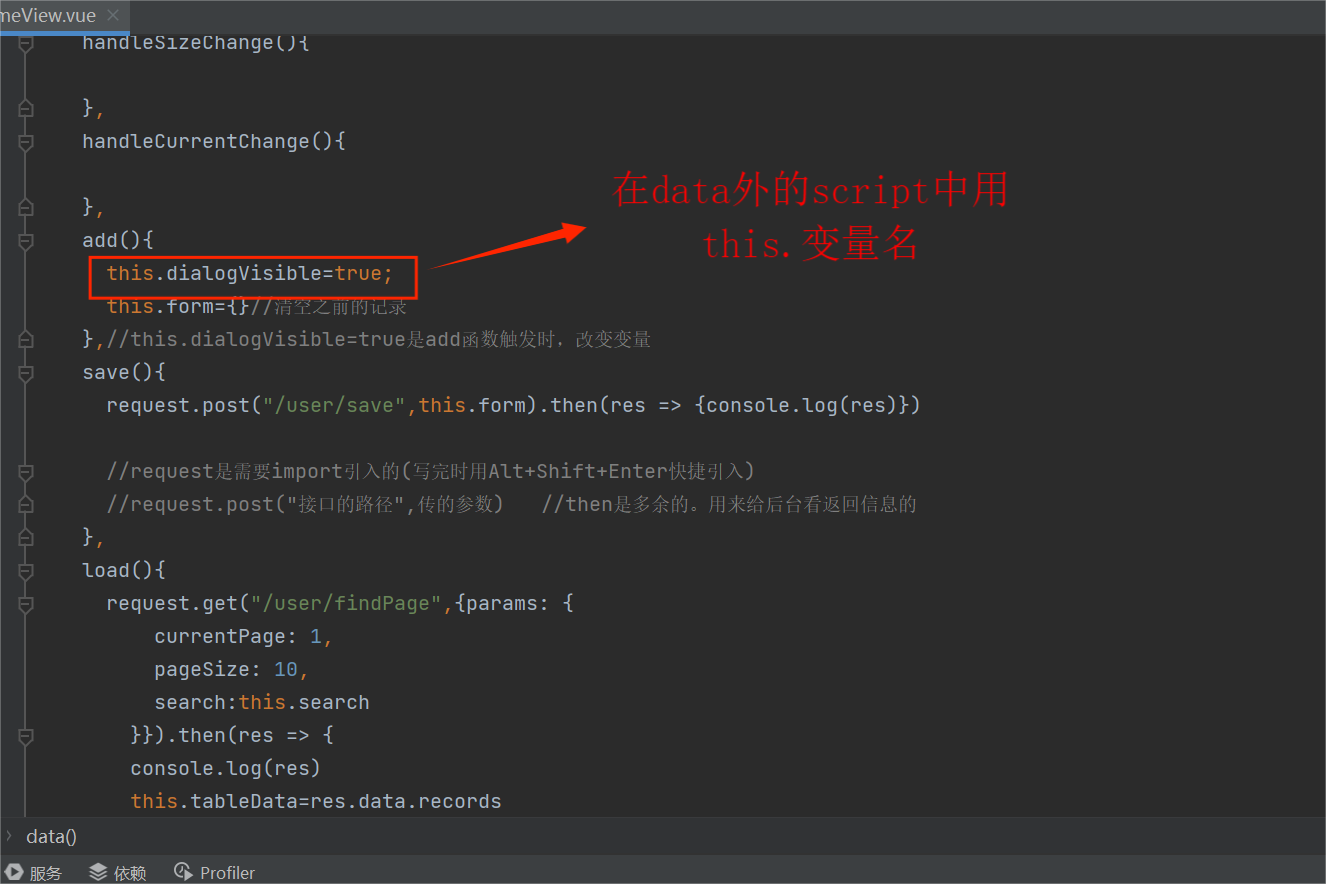
[springboot的接口的用法](note://WEB766ae4516c686313d3f62c0d6a41275b)

**简单页面变量的渲染**

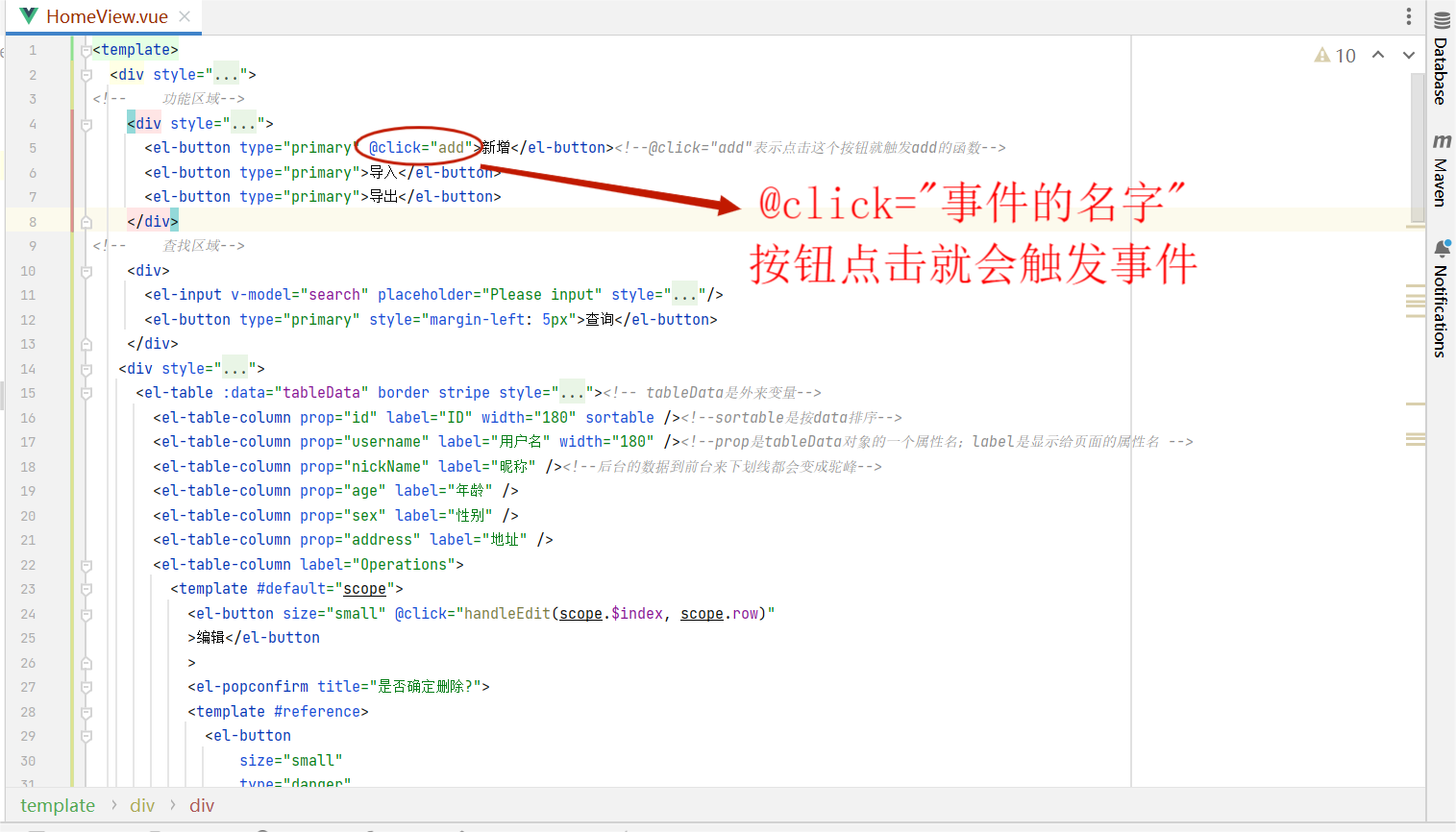


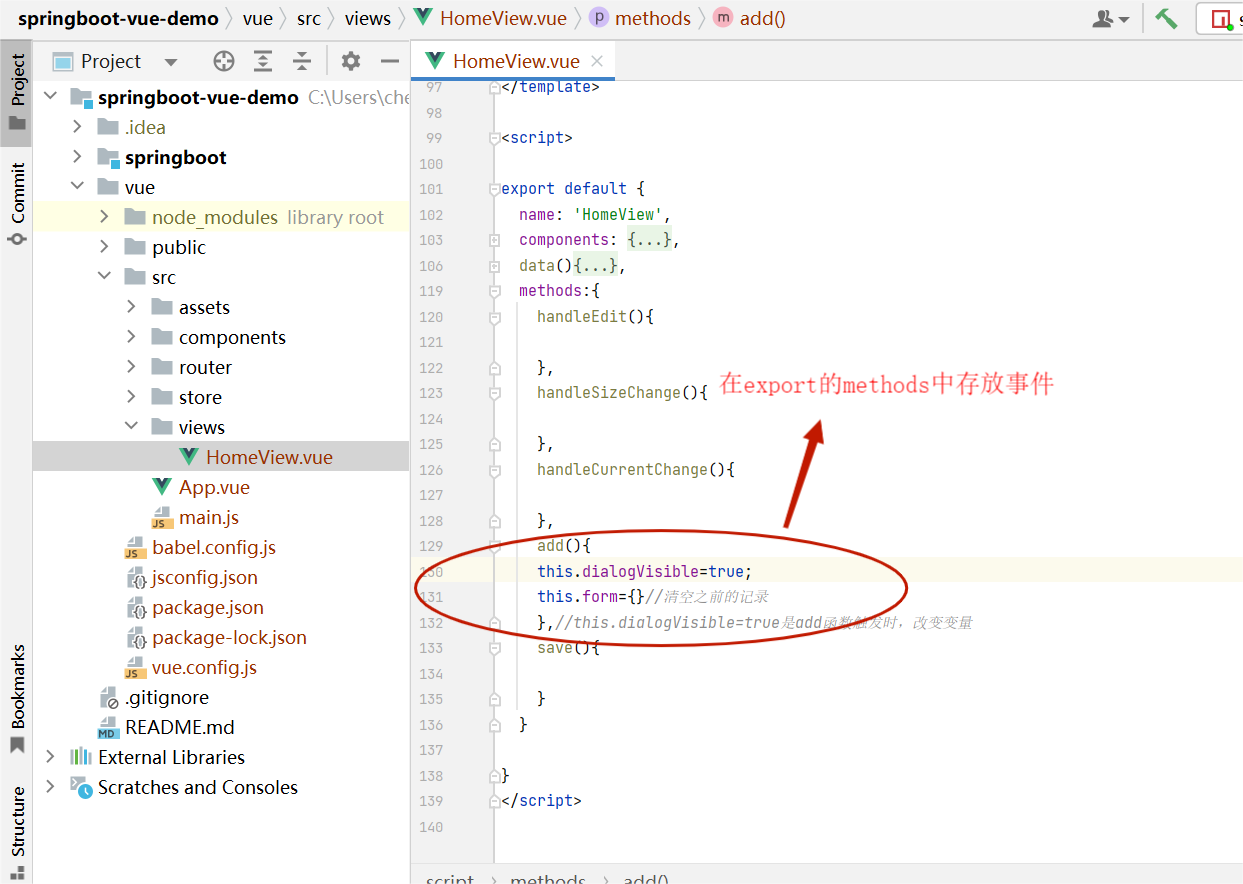


要用渲染的属性前面要有":"



**按钮的触发事件流程**





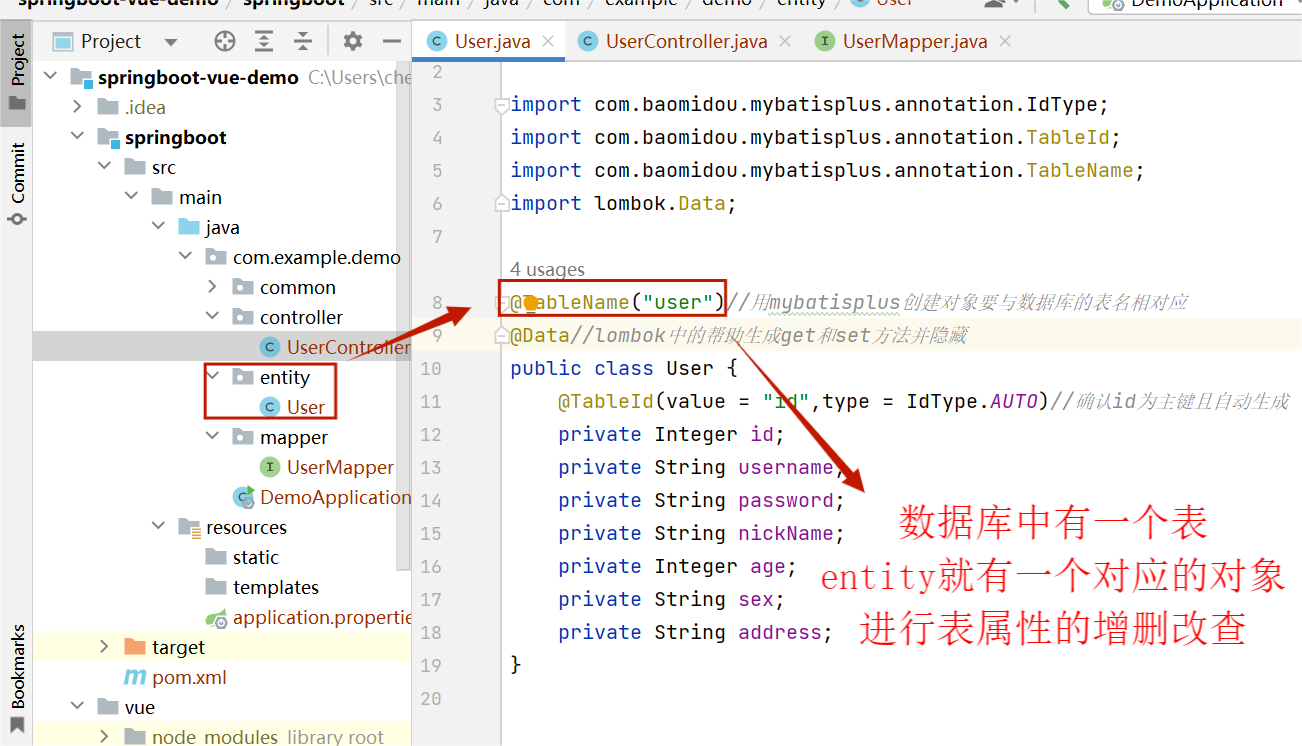
**将数据从前端到后端到数据库**

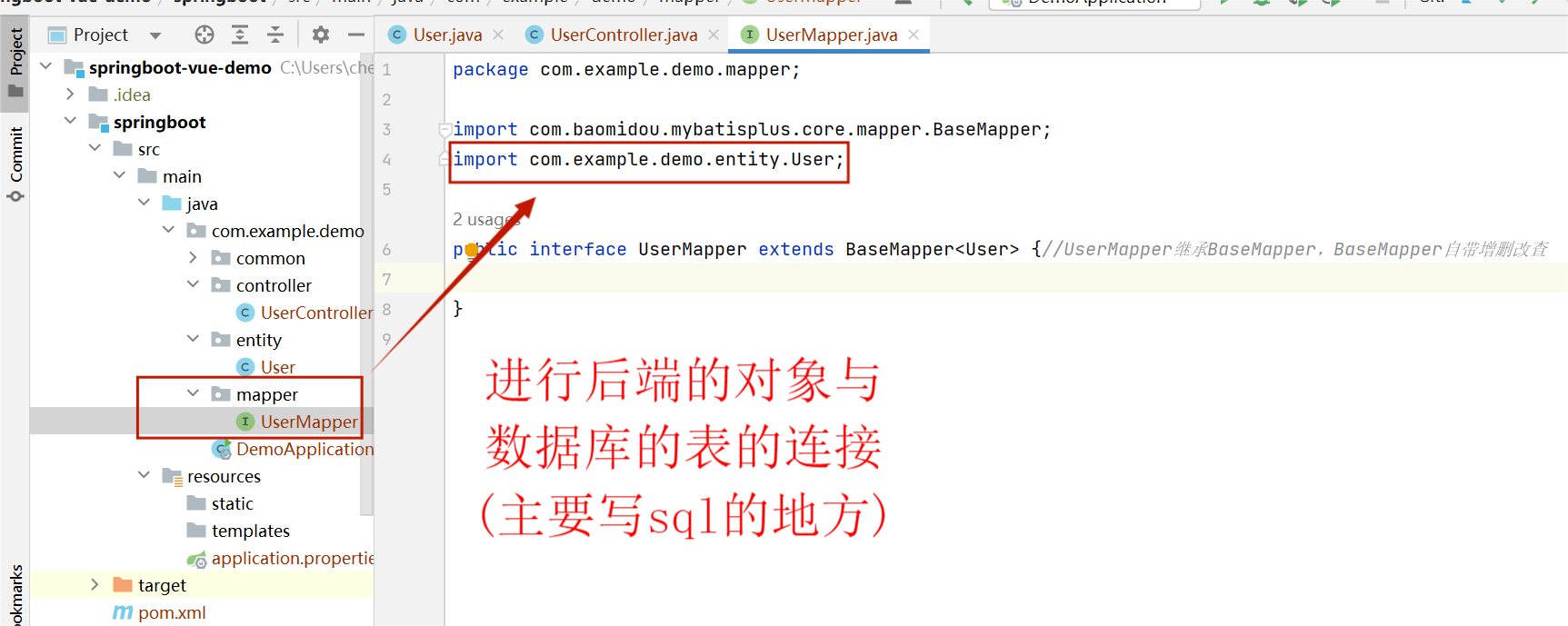
**前端的触发事件及其用法(前到后)**





**后端的controller和entity和mapper(后到库)**







**@PostMapper的前端的不同数据类型的处理(前端接收返回值时用request.post)多用于新增数据**

数据是对象(例如{id:'1',name:'2222'})就要用@RequestBody将json变为对象(例如:

@RequestBody ApplyObject applyObject)

数据是带有？的字符串(例如xxx/generatedData?id=1&name=222)就用@RequestParam 加上问号后面对应的字段名称(例如:

@RequestParam String id, @RequestParam String name)

**分页查询**

**前端页面的标签**

<!--分页-->

<div style="margin: 10px 0">

<el-pagination

v-model:currentPage="currentPage"

v-model:page-size="pageSize"

:page-sizes="[5, 10, 15, 20]"

:small="small"

:disabled="disabled"

:background="background"

layout="total, sizes, prev, pager, next, jumper"

:total="total"

@size-change="handleSizeChange"

@current-change="handleCurrentChange"

/><!--page-size一页多少条数据，page-sizes="[5, 10, 15, 20]"数据的间距，total="total"总页数-->

<!--@size-change="handleSizeChange"

@current-change="handleCurrentChange"

两个函数

-->

<el-dialog

v-model="dialogVisible"

title="提示"

width="30%"

>

<el-form :model="form" label-width="30%">

<el-form-item label="用户名">

<el-input v-model="form.username" style="width: 80%" /><!--:model="form"是绑定的对象变量；username是from的属性-->

</el-form-item>

<el-form-item label="昵称">

<el-input v-model="form.nickName" style="width: 80%"/>

</el-form-item>

<el-form-item label="年龄">

<el-input v-model="form.age" style="width: 80%"/>

</el-form-item>

<el-form-item label="性别">

<el-radio label="男" v-model="form.sex" size="large">男</el-radio>

<el-radio label="女" v-model="form.sex" size="large">女</el-radio>

<el-radio label="未知" v-model="form.sex" size="large">未知</el-radio>

</el-form-item>

<el-form-item label="地址">

<el-input type="textarea" v-model="form.address" style="width: 80%"/>

</el-form-item>

</el-form>

<template #footer>

<span class="dialog-footer">

<el-button @click="dialogVisible = false">取消</el-button>

<el-button type="primary" @click="save"

>确定</el-button

>

</span>

</template>

</el-dialog>

</div>

**前端script的变量和方法和启动伴随**

<script>

import request from "@/utils/request";

export default {

name: 'HomeView',

components: {

},

data(){

return{

form:{},

dialogVisible:false,

search:'',

currentPage:1,

total:0,

pageSize:10,

CQS:'新增',

tableData:[

]

}

},

created() {

this.load()

},

methods:{

handleEdit(row){

this.form=JSON.parse(JSON.stringify(row))//对数据进行深拷贝与其他form对象隔开

this.dialogVisible=true//打开弹窗

},

handleSizeChange(pageSize){//改变当前每页的个数触发

this.pageSize=pageSize

this.load()

},

handleCurrentChange(pageNum){//改变当前页码触发

this.currentPage=pageNum

this.load()

},

add(){

this.dialogVisible=true;

this.form={}//清空之前的记录

},//this.dialogVisible=true是add函数触发时，改变变量

save(){

if (this.form.id){//更新

request.put("/user/updata",this.form).then(res => {console.log(res)

if (res.code==='0') {//res.code是放回结果的一些性质,0就是成功

this.$message({type:"success",message:"更新成功"})//this.$message是自带的弹窗

}

else {

this.$message({type:"error",message:res.msg})

}

})

}

else {//新增

request.post("/user/save",this.form).then(res => {console.log(res)

if (res.code === '0') {//res.code是放回结果的一些性质,0就是成功

this.$message({type:"success",message:"新增成功"})//this.$message是自带的弹窗

}

else {

this.$message({type:"error",message:res.msg})

}

})

//request是需要import引入的(写完时用Alt+Shift+Enter快捷引入)

//request.post("接口的路径",传的参数) //then是多余的。用来给后台看返回信息的

}

this.load()

},

load(){

request.get("/user/findPage",{params: {

pageNum: this.currentPage,

pageSize: this.pageSize,

search:this.search

}}).then(res => {

console.log(res)

this.tableData=res.data.records

this.total=res.data.total

})

}

}

}

</script>

**后端分页配置**

**要有common的配置文件**

链接：<https://pan.baidu.com/s/12SJD1h6AAv7TShglYuLkfg>

提取码：1111

**application.properties的数据库要连好**

server.port = 9090

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/springboot-vue?useUnicode=true&characterEncoding=utf8

spring.datasource.username=root

spring.datasource.password=Cqs123456789

**pom.xml的依赖要下完**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.7.3</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.example</groupId>

<artifactId>demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>demo</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>11</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

<version>2.2.2</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-boot-starter</artifactId>

<version>3.5.1</version>

</dependency>

<dependency>

<groupId>cn.hutool</groupId>

<artifactId>hutool-all</artifactId>

<version>5.8.5</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<excludes>

<exclude>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</exclude>

</excludes>

</configuration>

</plugin>

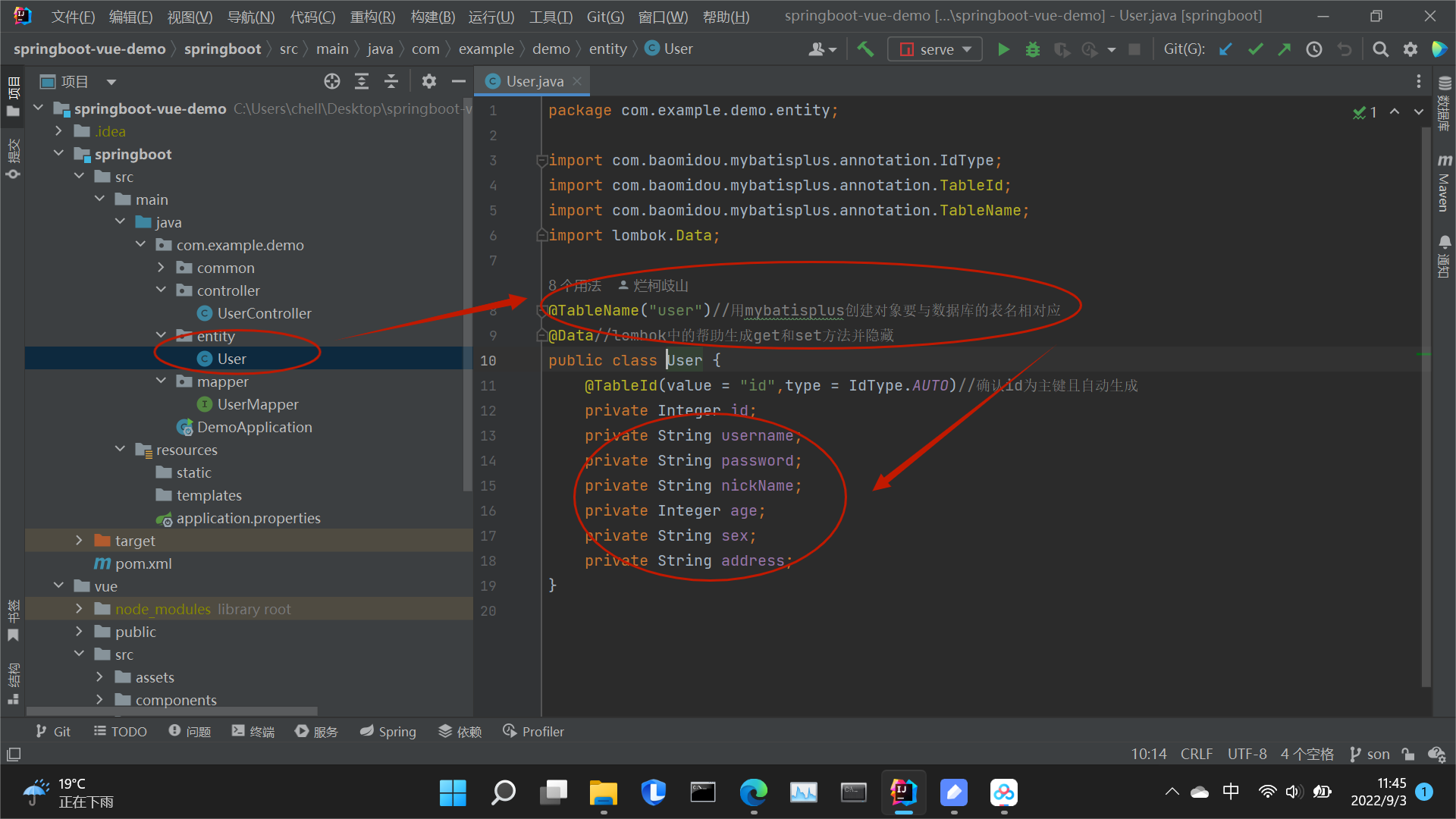
</plugins>

</build>

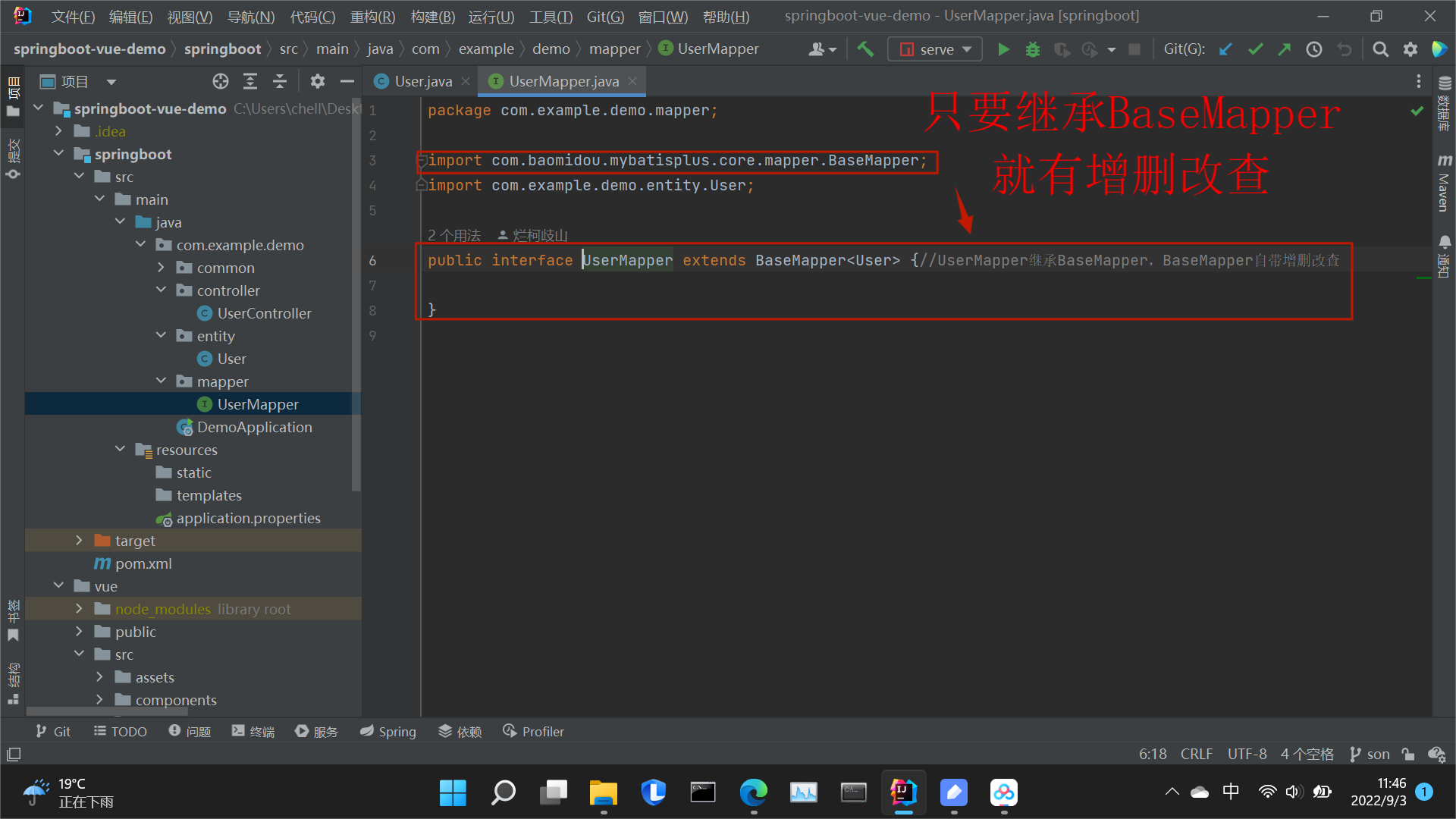
</project>

**后端对象及其控制器和实现类**

**对象(对应表的)**



**对象对应的实现类**



**对象对应的控制器**

package com.example.demo.controller;

import cn.hutool.core.util.StrUtil;

import com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;

import com.baomidou.mybatisplus.core.toolkit.Wrappers;

import com.baomidou.mybatisplus.extension.plugins.pagination.Page;

import com.example.demo.common.Result;

import com.example.demo.entity.User;

import com.example.demo.mapper.UserMapper;

import org.springframework.web.bind.annotation.\*;

import javax.annotation.Resource;

import java.sql.Wrapper;

//@RestController的意思是表示自己为返回json的控制器

@RestController

@RequestMapping("/user")//@RequestMapping是usercontroller统一的路由

public class UserController {

@Resource//引入Mapper的userMapper

UserMapper userMapper;

//注册方法

@PostMapping(value="/save")//定义post的接口

public Result<?> save(@RequestBody User user){//RequestBody 可以将前台传来的json转化为User对象

//Result<?>的问号表示所以类型的数据都可以接受

if (user.getPassword()==null) {

user.setPassword("123456");

}

userMapper.insert(user);

return Result.success();//放回信息

}

//分页方法

@GetMapping(value = "/findPage")

public Result<?> findPage(@RequestParam(defaultValue = "1") Integer pageNum,@RequestParam(defaultValue = "10") Integer pageSize,@RequestParam(defaultValue = "") String search){

//pageNum是当前页;pageSize是一页有多少条数据;search是查询的关键字(defaultValue是设置默认值)

LambdaQueryWrapper<User> wrapper=Wrappers.<User>lambdaQuery();//sql条件的执行对象

if (StrUtil.isNotBlank(search)){//StrUtil.isNotBlank(search)是hutool依赖用来帮助判断参数是否为空的

wrapper.like(User::getNickName,search);

}

Page<User> userPage= userMapper.selectPage(new Page<>(pageNum,pageSize), wrapper);

//selectPage需要Page对象和sql执行对象

//new Page<>(pageNum,pageSize)//分页对象

//Wrappers.<User>lambdaQuery().like(User::getNickName,search)//sql执行对象

// User::getNickName是User.getNickName的变形

return Result.success(userPage);//Result.success(变量名)//用返回变量

}

}

**@GetMapper的前端的不同数据类型的处理(前端接收返回值时用request.get加params)多用于查询数据**

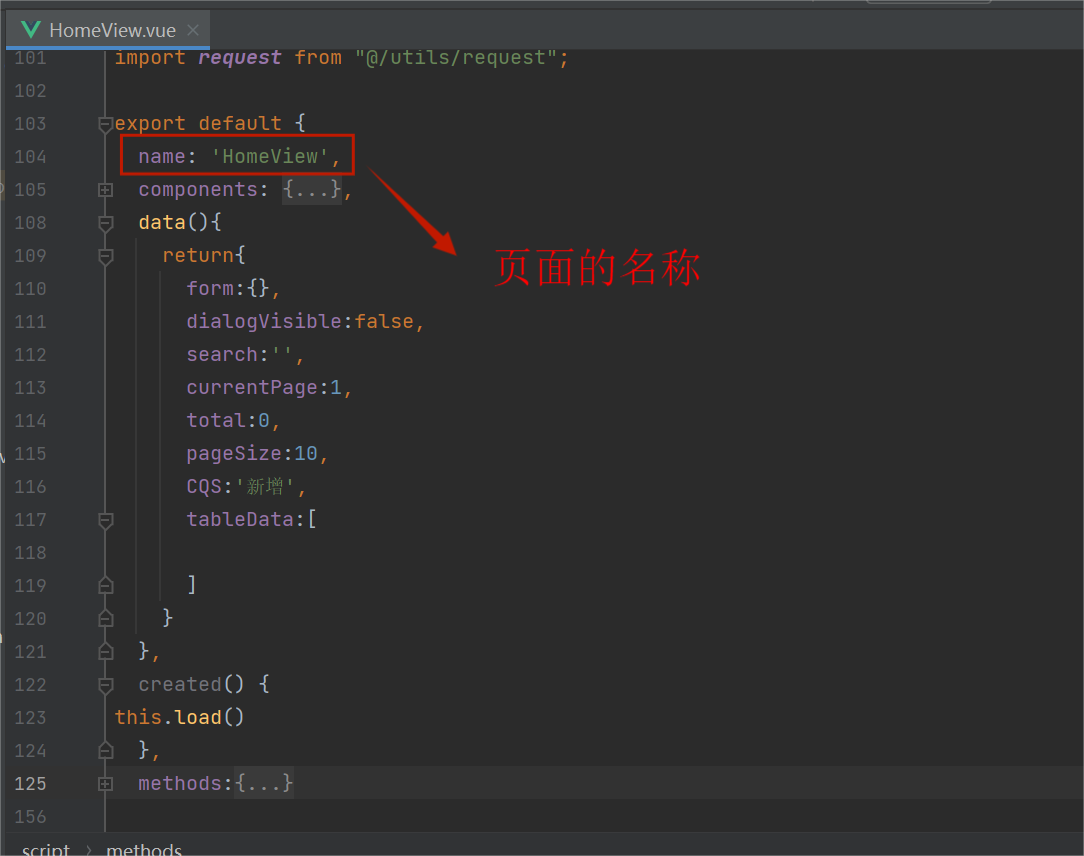
数据是对象(例如{id:'1',name:'2222'})就不用@RequestBody(例如:

ApplyObject applyObject)

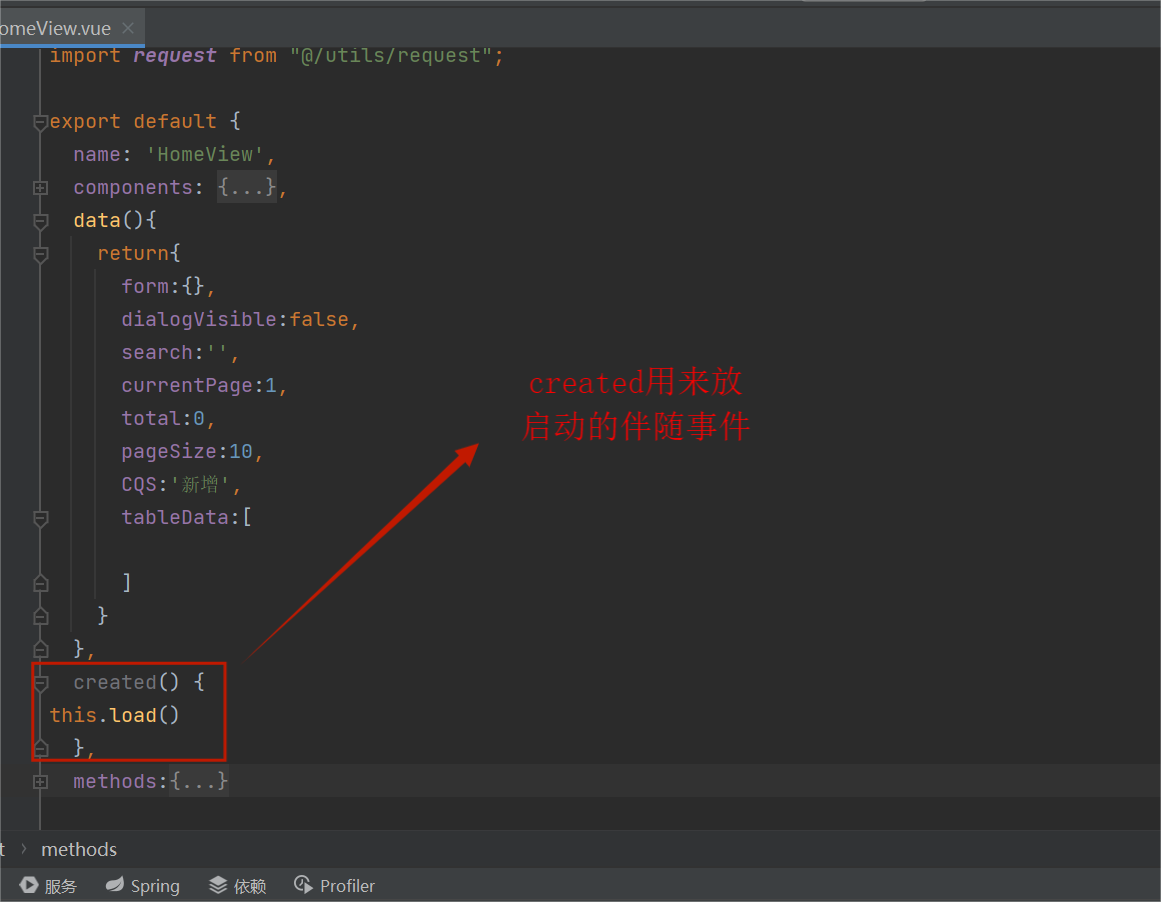
数据是带有？的字符串(例如xxx/generatedData?id=1&name=222)就用@RequestParam 加上问号后面对应的字段名称(例如:

@RequestParam String id, @RequestParam String name)

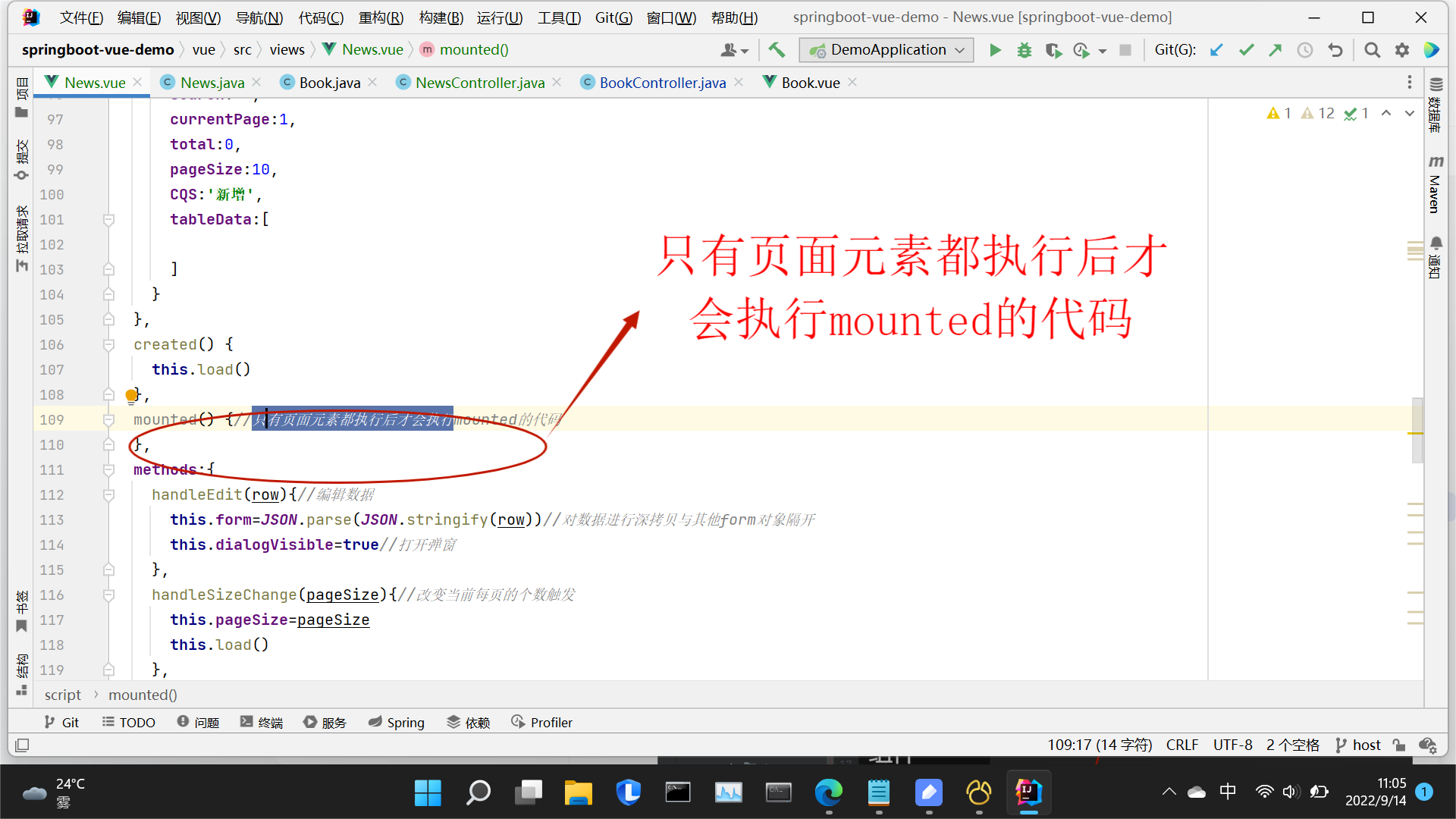
**页面名称的定义**



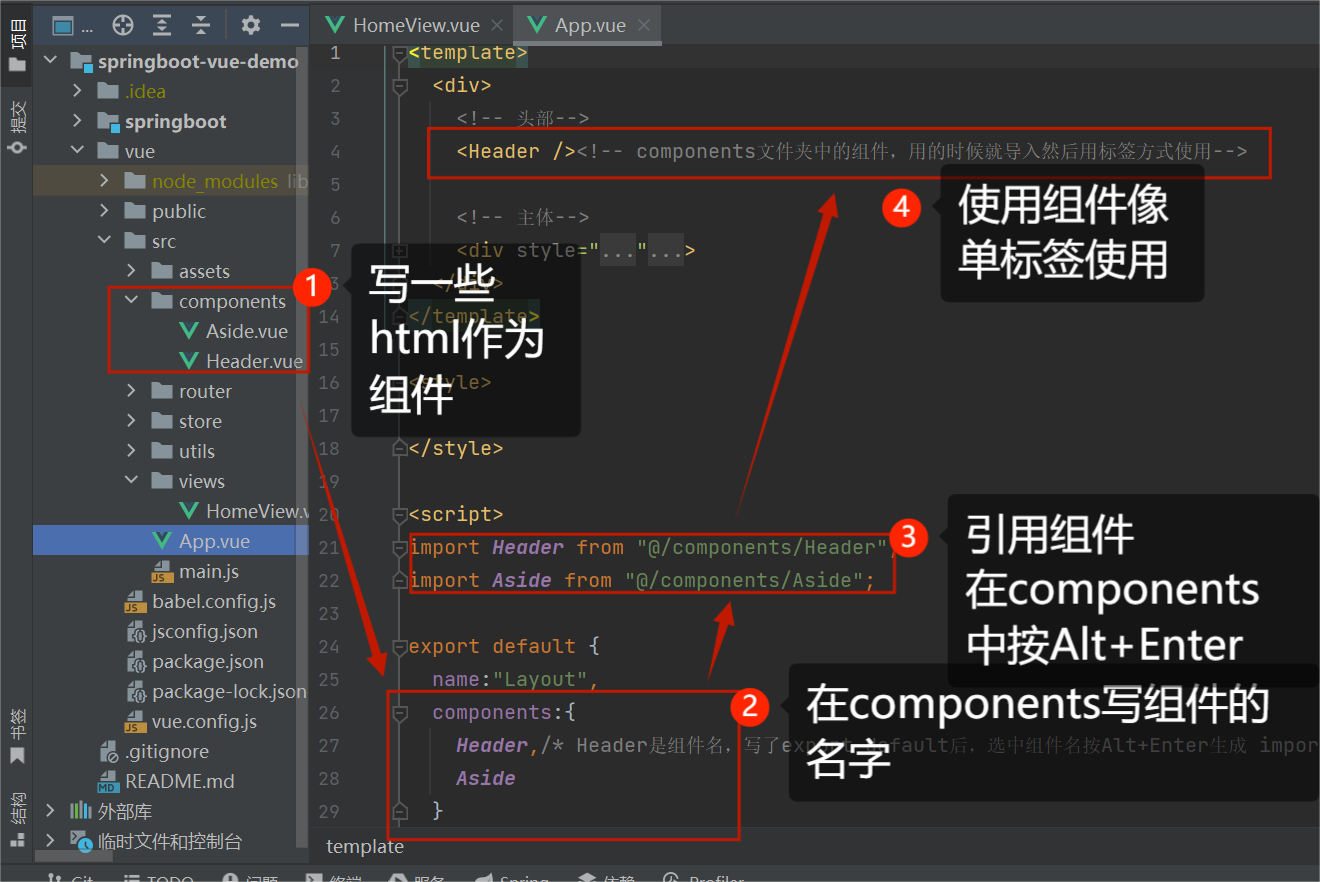
**页面启动的伴随操作的定义**



**只有页面元素都执行后才会执行**



**页面调用的组件(自定义标签组)**



**编辑页面数据同步数据库**

**后端控制器加的接口**

@PutMapping(value="/updata")//定义put的接口

public Result<?> updata(@RequestBody User user){//RequestBody 可以将前台传来的json转化为User对象

//Result<?>的问号表示所以类型的数据都可以接受

userMapper.updateById(user);

return Result.success();//放回信息

}

**前端页面的事件**

handleEdit(row){//编辑数据

this.form=JSON.parse(JSON.stringify(row))//对数据进行深拷贝与其他form对象隔开

this.dialogVisible=true//打开弹窗

},

save(){

if (this.form.id){//更新

request.put("/user/updata",this.form).then(res => {console.log(res)//put是@PutMapping对应

if (res.code==='0') {//res.code是放回结果的一些性质,0就是成功

this.$message({type:"success",message:"更新成功"})//this.$message是自带的弹窗

}

else {

this.$message({type:"error",message:res.msg})

}

})

}

else {//新增

request.post("/user/save",this.form).then(res => {console.log(res)

if (res.code === '0') {//res.code是放回结果的一些性质,0就是成功

this.$message({type:"success",message:"新增成功"})//this.$message是自带的弹窗

}

else {

this.$message({type:"error",message:res.msg})

}

})

//request是需要import引入的(写完时用Alt+Shift+Enter快捷引入)

//request.post("接口的路径",传的参数) //then是多余的。用来给后台看返回信息的

}

this.load()

},load(){

request.get("/user/findPage",{params: {

currentPage: this.currentPage,

pageSize: this.pageSize,

search:this.search

}}).then(res => {

console.log(res)

this.tableData=res.data.records

this.total=res.data.total

})

},

**前端标签**

<el-button size="small" @click="handleEdit(scope.row)"

>编辑</el-button

>

**@PutMapper对应put(多用于修改数据)**

与post类似

**删除数据**

**后端控制器加的接口**

@DeleteMapping(value="/delUser/{id}")///delUser/{id}/{aa}可以同时传入两个数据

public Result<?> delUser(@PathVariable Long id){//@PathVariable花括号中的字符串变为对象,搭配@DeleteMapping

//Result<?>的问号表示所以类型的数据都可以接受

userMapper.deleteById(id);

return Result.success();//放回信息

}

**前端页面的事件**

load(){

request.get("/user/findPage",{params: {

currentPage: this.currentPage,

pageSize: this.pageSize,

search:this.search

}}).then(res => {//res表示接口返回信息

console.log(res)

this.tableData=res.data.records

this.total=res.data.total

})

},

handleDelete(id){

console.log(id)

request.delete("/user/delUser/"+id).then(res=>{

if (res.code === '0') {//res.code是放回结果的一些性质,0就是成功

this.$message({type:"success",message:"删除成功"})//this.$message是自带的弹窗

}

else {

this.$message({type:"error",message:res.msg})

}

})Z

this.load()

}

**前端的标签**

<el-popconfirm title="是否确定删除?" @confirm="handleDelete(scope.row.id)"><!--气泡绑定按钮的事件用@confirm-->

<template #reference>

<el-button

size="small"

type="danger"

>删除</el-button

>

</template>

</el-popconfirm>

**@DeleteMapping(value="/delUser/{id}")///delUser/{id}/{aa}可以同时传入两个数据(多用于删除数据)**

public Result<?> delUser(@PathVariable Long id)//@PathVariable花括号中的字符串变为对象,搭配@DeleteMapping

前端事件得用request.delete("/user/delUser/"+id)

**页面跳转**

**事件方式**

this.$router.push("/")//进行页面跳转(到"/")

**标签属性方式(一)**

<el-menu-item index="data" :route="{path:'/'}">数据管理</el-menu-item>

<!-- :route="{path:'/'}就是页面跳转的属性-->

**标签属性方式(二)**

<el-menu

style="width: 200px;min-height: calc(100hv - 50px)"

default-active="user"

router

class="el-menu-vertical">

<!-- router和el-menu-item的index组合为/index值的路径跳转-->

<el-sub-menu index="1">

<template #title>系统管理</template>

<el-menu-item index="user" >用户管理</el-menu-item>

</el-sub-menu>

<el-menu-item index="data">数据管理</el-menu-item>

</el-menu>

**事件触发弹窗**

//事件成功(type:"success"),也可以(type:"error")表示错误

this.$message({type:"success",message:"登录成功"})//this.$message是弹窗组件

//弹窗信息(message:"登录成功")

//这样可以进行"/user/login"的事件成败信息的弹窗

request.post("/user/login",this.form).then(res=>{

if (res.code === '0') {//res.code是放回结果的一些性质,0就是成功

this.$message({type:"success",message:"登录成功"})//this.$message是

this.$router.push("/")//登录成功后进行页面跳转

}

else {

this.$message({type:"error",message:res.msg})

}

})

**request语句的结构**

**登录**

**前端页面**

<template>

<div style="width: 100%;height: 100vh;background-color: darkslateblue;overflow: hidden">

<div style="width: 400px;margin: 150px auto">

<div style="color: #cccccc;font-size: 30px;text-align: center;padding: 30px 0">欢迎登录</div>

<el-form :model="form" size="normal" :rules="rules" ref="form" >

<el-form-item prop="username">

<el-input v-model="form.username" :prefix-icon="Avatar" />

</el-form-item>

<el-form-item prop="password">

<el-input v-model="form.password" show-password :prefix-icon="Lock"/>

</el-form-item>

<el-form-item >

<el-button style="width: 100%;" type="primary" @click="login">登 录</el-button>

</el-form-item>

</el-form>

</div>

</div>

</template>

<script>

import { Avatar,Lock} from '@element-plus/icons-vue'

import request from "@/utils/request";

// 统一导入el-icon图标

export default {

name: "Login",

data(){

return{

form:{},

rules:{

username:[

{required:true,message:"请输入用户名",trigger:"blur"},

],

password:[

{required:true,message:"请输入密码",trigger:"blur"},

]

}

}

},

setup(){

return {

Avatar,

Lock

}

},

methods:{

login(){

this.$refs['form'].validate((valid) => {//表格标签的ref属性的验证合法(就是密码和账号同时输入)

if (valid) {

request.post("/user/login",this.form).then(res=>{

if (res.code === '0') {//res.code是放回结果的一些性质,0就是成功

this.$message({type:"success",message:"登录成功"})//this.$message是

this.$router.push("/")//登录成功后进行页面跳转

}

else {

this.$message({type:"error",message:res.msg})

}

})

}

})

}

}

}

</script>

**后端端口**

public Result<?> login(@RequestBody User user){//RequestBody 可以将前台传来的json转化为User对象

//Result<?>的问号表示所以类型的数据都可以接受

User res = userMapper.selectOne(Wrappers.<User>lambdaQuery().eq(User::getUsername,user.getUsername()).eq(User::getPassword,user.getPassword()));

if (res == null){

return Result.error("-1","用户名或密码错误");//错误信息的返回

}

return Result.success();//放回信息

}

**退出系统**

<el-dropdown-item @click="$router.push('/Login')">退出系统</el-dropdown-item>

<!-- @click="$router.push('/Login')" 就是点击这个按钮就跳转页面-->

**注册及其验证**

**前端页面**

<template>

<div style="width: 100%;height: 100vh;background-color: darkslateblue;overflow: hidden">

<div style="width: 400px;margin: 150px auto">

<div style="color: #cccccc;font-size: 30px;text-align: center;padding: 30px 0">欢迎注册</div>

<el-form :model="form" size="normal" :rules="rules" ref="form">

<el-form-item prop="username">

<el-input v-model="form.username" :prefix-icon="Avatar" />

</el-form-item>

<el-form-item prop="password">

<el-input v-model="form.password" show-password :prefix-icon="Lock"/>

</el-form-item>

<el-form-item prop="confirm">

<el-input v-model="form.confirm" show-password :prefix-icon="Lock"/>

</el-form-item>

<el-form-item >

<el-button style="width: 100%;" type="primary" @click="register">注 册</el-button>

</el-form-item>

</el-form>

</div>

</div>

</template>

<script>

import { Avatar,Lock} from '@element-plus/icons-vue'

import request from "@/utils/request";

// 统一导入el-icon图标

export default {

name: "Register",

data(){

return{

form:{},

rules:{

username:[

{required:true,message:"请输入用户名",trigger:"blur"},

],

password:[

{required:true,message:"请输入密码",trigger:"blur"},

],

confirm:[

{required:true,message:"请确认密码",trigger:"blur"},

]

}

}

},

setup(){

return {

Avatar,

Lock

}

},

methods:{

register(){

if (this.form.password!==this.form.confirm){

this.$message({type:"error",message:"两次密码输入不一致"})

return

}

this.$refs['form'].validate((valid) => {//表格标签的ref属性的验证合法(就是密码和账号同时输入)

if (valid) {

request.post("/user/register",this.form).then(res=>{

if (res.code === '0') {//res.code是放回结果的一些性质,0就是成功

this.$message({type:"success",message:"注册成功"})//this.$message是

this.$router.push("/Login")//登录成功后进行页面跳转

}

else {

this.$message({type:"error",message:res.msg})

}

})

}})

}

}

}

</script>

**后端端口**

@PostMapping(value="/register")//定义post的接口

public Result<?> register(@RequestBody User user){//RequestBody 可以将前台传来的json转化为User对象

//Result<?>的问号表示所以类型的数据都可以接受

User res = userMapper.selectOne(Wrappers.<User>lambdaQuery().eq(User::getUsername,user.getUsername()));

if (res != null){

return Result.error("-1","用户名重复");//错误信息的返回

}

if(user.getPassword()==null){

user.setPassword("123456");

}

userMapper.insert(user);

return Result.success();//放回信息

}

**登录拦截与信息缓存**

**前端缓存信息**

sessionStorage.setItem("user", JSON.stringify(res.data)) // 缓存用户信息

**request.js**

// 取出sessionStorage里面缓存的用户信息

let userJson = sessionStorage.getItem("user")

if (!whiteUrls.includes(config.url)) { // 校验请求白名单

if(!userJson) {

router.push("/login")

} else {

let user = JSON.parse(userJson);

// config.headers['token'] = user.token; // 设置请求头

}

}

**文件上传**

**前端**

<el-form-item label="封面">

<el-upload action="http://localhost:9090/files/upload" :on-success="filesUploadSuccess">

<el-button type="primary">点击上传</el-button>

</el-upload>

</el-form-item>

**后端**

@Value("${server.port}")//获取项目的端口号

private String port;

private static final String ip="http://localhost";//获取项目的IP

@PostMapping("/upload")

public Result<?> upload(MultipartFile file) throws IOException {

String originalFilename=file.getOriginalFilename();//获取文件名

String flag = IdUtil.fastSimpleUUID();// 定义文件的唯一标识（前缀）

String rootFilePath=System.getProperty("user.id")+"/springboot/src/main/resources/files"+flag + "\_" +originalFilename;//获取files文件夹的绝对路径

FileUtil.writeBytes(file.getBytes(),rootFilePath);//写入到上传的路径

return Result.success( ip + ":" + port + "/files/" + flag); // 返回结果 url

**文件下载**

**前端**

<el-table-column

label="封面">

<template #default="scope">

<el-image

style="width: 100px; height: 100px"

:src="scope.row.cover"

:preview-src-list="[scope.row.cover]">

</el-image>

</template>

</el-table-column>

**后端**

@GetMapping("/{flag}")

public void getFiles(@PathVariable String flag, HttpServletResponse response){//由于文件的下载是通过流的形式，不需要Result返回

OutputStream os; // 新建一个输出流对象

String basePath = System.getProperty("user.dir") + "/springboot/src/main/resources/files/"; // 定于文件上传的根路径

List<String> fileNames = FileUtil.listFileNames(basePath); // 获取所有的文件名称

String fileName = fileNames.stream().filter(name -> name.contains(flag)).findAny().orElse(""); // 找到跟参数一致的文件

try {

if (StrUtil.isNotEmpty(fileName)) {

response.addHeader("Content-Disposition", "attachment;filename=" + URLEncoder.encode(fileName, "UTF-8"));

response.setContentType("application/octet-stream");

byte[] bytes = FileUtil.readBytes(basePath + fileName); // 通过文件的路径读取文件字节流

os = response.getOutputStream(); // 通过输出流返回文件

os.write(bytes);

os.flush();

os.close();

}

} catch (Exception e) {

System.out.println("文件下载失败");

}

}

**异步元素的防爆错机制**

this.$nextTick(()=>{//是否有异步元素的判断

// 里面写异步元素

})

**信息的缓存session**

**前端**

// 缓存用户信息

sessionStorage.setItem("user", JSON.stringify(res.data))

//获取缓存

let userStr=sessionStorage.getItem("user")||"{}"

let user=JSON.parse(userStr)

this.nickName=user.nickName

**后端**

return Result.success(res);//放回信息

**管理员和普通用户的权限控制(弱安全性)**

script中

created() {

let userStr=sessionStorage.getItem("user")||"{}"

this.user=JSON.parse(userStr)

}

标签中(v-if用来判断角色权限)

<el-sub-menu index="1" v-if="user.role === 1">