

上海海洋大学2025项目实训

前后端分离开发

前端(app):

VSCode工具：前端代码编辑工具。 免费，官网下载。 【自己安装】

开发环境：

nvm工具：下载和管理**Node**软件。 **node**就是前端框架运行的环境。

nvm作用就是可以在电脑上安装不同版本**node**，随意切换。

技术：**Vue2**、**Vue3**框架。

vue的版本不同，所需要**node**版本也不同。

运行：

浏览器 **Chrome** 安装插件。

Edge 安装插件（微软在线商店）。

后端（服务器程序）：

Idea2023版本 【自己安装破解】

开发环境：

JDK8

maven3.8.8 管理项目和项目中依赖(jar)。

数据库：

MySQL5 或 **MySQL8** 【自己安装，**root**用户密码记住】

Navicat Premium 17 数据库可视化的工具（破解）。

测试：

ApiFox 【自己安装】

nvm工具使用

作用：**nvm**可以帮助我们安装和管理**node**软件。

安装

下载**nvm**工具。安装 **D:\tools\nvm** 目录。

安装完毕实际路径：**D:\tools\nvm\nvm**

配置

步骤1:

首先在**D:\tools** 路径创建一个**nodejs**文件夹。指定**nvm**工具安装**node**到该文件夹下面。

创建完毕目录：**D:\tools\nodejs**

步骤2:

查看**nvm**环境变量（正常情况安装**nvm**自动配置环境变量）

NVM_HOME: **D:\tools\nvm** **nvm**安装的根目录

NVM_SYMLINK: **D:\tools\nodejs**

步骤3:

修改**nvm**软件配置参数，打开**nvm**安装目录，打开**settings.txt**文件，配置以下内容：

root:**D:\tools\nvm**

path:**D:\tools\nodejs**

node_mirror:<https://npmmirror.com/mirrors/node/>

```
npm_mirror:https://npmmirror.com/mirrors/npm/
```

步骤4

测试配置是否成功

启动Cmd终端，视窗+S 搜索cmd，以管理员的身份打开：

输入命令测试：

```
nvm version    --查询nvm版本
1.1.10
```

nvm安装NodeJS

如果之前电脑安装过NodeJS，现在想使用nvm，就必须先卸载之前安装nodeJS

--查看nvm镜像服务器上有哪些NodeJS版本。

```
>nvm list available
```

--安装指定版本nodeJS

```
>nvm install 18.19.0
```

--查看电脑已经安装的版本

```
>nvm list
```

--切换版本命令

```
>nvm use 18.19.0
```

--卸载的命令

```
>nvm uninstall 18.19.0
```

node和npm配置

node前端程序运行环境，node安装成功自带一个npm工具。

npm前端项目构建、相关插件的依赖库的工具。

使用命令查看 node版本。

```
>node -v
v18.19.0
```

node工具安装完毕，自动在node内部安装一个npm工具。

--查看当前npm版本

```
>npm -v
10.2.3
```

--查看npm默认的配置信息

```
>npm config list
```

配置npm信息

配置信息 实质在上 C:\Users\byterain\.npmrc 文件中存储。

所以，如果配置错误，可以将该文件删除，重新配置。

步骤1:

设置npm文件

在D:\tools\nvm\v18.19.0目录下，创建2个文件夹：

npm_cache 安装插件时所需要的缓存文件夹

npm_modules 安装插件所存放的目录

步骤2:

启动cmd终端, 配置新建的2个文件夹到npm。

```
npm config set prefix "D:\tools\nvm\v18.19.0\npm_modules"
```

```
npm config set cache "D:\tools\nvm\v18.19.0\npm_cache"
```

步骤3:

设置npm下载的服务器地址为国内的镜像地址

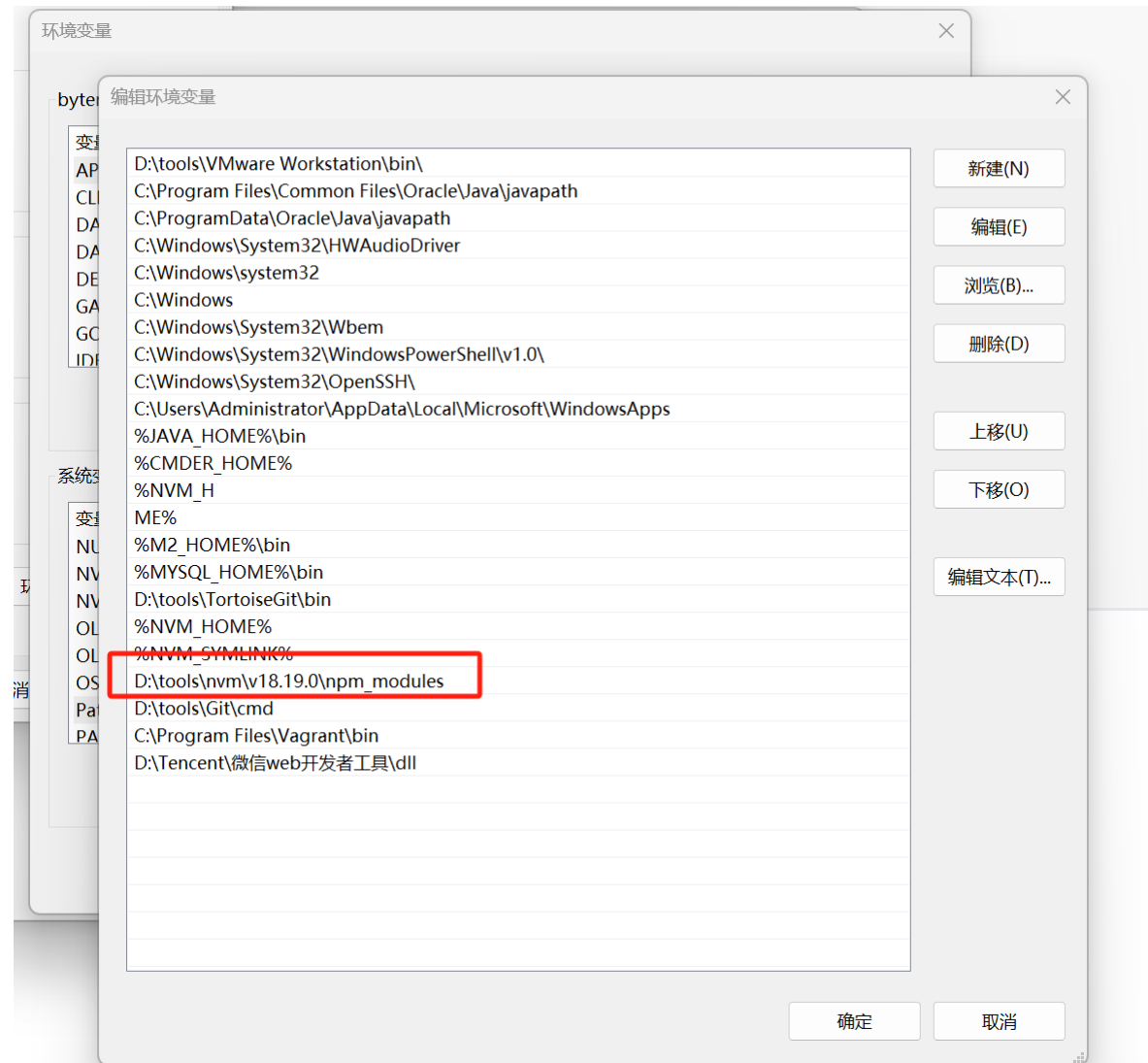
```
npm config set registry http://registry.npmirror.com
```

步骤4:

```
npm config list
```

配置npm环境变量

添加 D:\tools\nvm\v18.19.0\npm_modules 到系统Path环境变量中。



使用npm创建vue项目

安装插件 脚手架(快速搭建Vue项目结构)。

-- npm安装插件命令

```
npm install 插件@版本
```

-- 安装vue脚手架插件

```
>npm install -g @vue/cli
```

```
--测试查看脚手架插件是否安装成功
>vue -V      --大写V
@vue/cli 5.0.8
```

创建Vue项目

切换到项目存储目录，使用命令创建vue项目

```
vue create 项目名（全部小写）
```

使用Vscode编辑器打开项目，在VScode中启动终端：Ctrl + `

```
-- 启动项目
D:\VueProjects\mall_vue_021> npm run serve
-- 注意：必须在项目文件夹路径下执行
```

终止终端服务运行

快捷键 Ctrl+c 输入y

Vue项目结构

```
mall_vue_021
|- node_modules 项目运行所需的插件库
|- public
|  |- favicon.ico 浏览器选项卡上图标
|  |- index.html 首页（单页面应用）
|- src 代码文件夹
|  |- assets 静态资源(图片)
|  |- components 公共组件文件夹
|  |- router 路由( 切换页面 )
|  |- views 所有视图界面( 视图组件)
|  |- App.vue 主组件
|  |- main.js 入口文件(项目代码从入口文件开始执行)
|- package.json 拆解库的配置文件,内部所有的配置数据都是以json形式配置
|- package-lock.json 定义各插件最高版本上限
|- vue.config.js 项目的配置文件，修改端口号，跨域。
```

Vue项目中 所有的组件文件后缀名都是.vue文件。

json对象

Javascript代码定义一个对象（数据的集合体 -- 变量和方法）。

```
let obj = { 属性:值,属性:值,属性:值 };
对象的属性值 可以是JS中任意类型： 数字、字符串、逻辑类型、方法(函数)、对象、数组...

<script>
  let stu = {
    name:"byterain",
    age:22,
    phone:'13388990099',
    getNameLength:function(){
      return this.name.length;
    }
  }
</script>
```

```
    },
    address: {
      code: '030006',
      info: '上海市静安区南京西路1号'
    },
    language: ["Java", "C", "C++"]
  };

  console.log(stu.name);
  console.log(stu.getNameLength());
  console.log(stu.address.info);
  console.log(stu.language);
</script>
```

vue2页面代码结构

- 1、页面内容--html代码
- 2、页面代码--Javascript代码
- 3、页面样式--css样式

```
<template>
  <div>

  </div>
</template>

<script>
export default {
  data() {
    return {
    };
  },
  methods: {
  },
};
</script>

<style scoped>
</style>
```

定义路由规则

路由就是一对key-value的对应关系，key就是访问的路径，value就是访问的组件界面。

配置路由需要编辑 router/index.js文件。

```
import { createRouter, createWebHashHistory } from 'vue-router'
//所有需要路由的组件界面 导入 @符合表示 src根路径
import Home from '@views/Home.vue'
import Login from '@views/Login.vue'    //ES6 模块化编程

//定义路由规则
const routes = [
  //一个路由对象，就是一个组件界面的对应关系
  {
    path: '/',
```

```

    name: 'home',
    component: Home
  },
  {
    path: '/login',
    name: 'login',
    component: Login
  }
]

//创建路由对象
const router = createRouter({
  history: createWebHashHistory(),
  //路由规则   routes:routes 属性名和属性值是同名，可以简写 routes
  routes
})

export default router
/**
  export default 默认对象

  其它代码中
  import 默认对象 导入
  */

```

编辑main.js文件，导入路由对象

```

import { createApp } from 'vue'
import App from './App.vue'
// . 当前路径， ./router/index.js 如果导入index.js文件，可以省略
import router from './router'

createApp(App).use(router).mount('#app')

//# 代码 id   #app代表 就是  id="app"

```

ES6

箭头函数

```

//ES5 定义函数
function f(v){
  return v;
}

//ES6
const f = (v)=>{
  return v;
}

//如果箭头函数的方法体中，只有一句代码，并且是return，那么可以简写
const f = (v) => v;
//(v)就是函数参数，如果只有一个参数，那么()小括号可以省略
const f=v=>v;

const sum = (a,b)=>a+b;

function sum(a,b){

```

```
    return a+b;
}
```

模版字符串

```
let a = "Hello";
let b = "Vue";
//Hello # Vue
//+ 运算符，2边是数字进行数学运算，如果+2边任一方是字符串类型，进行字符串拼接。

let str = a+" # "+b;

let str = `${a} # ${b}`;
```

Web存储

本地存储: LocalStorage 关闭浏览器，存储数据依旧存在。

会话存储: SessionStorage 关闭浏览器，存储内容就会丢失。

都是通过 name-value 形式进行存储的。

模块化实现

javascript没有模块编程概念。

将一段代码独立为一个.js文件。
页面导入使用js文件
`<script src="js文件路径"></script>`

模块化 通过2个命令构成: export和import。

export 命令用于规定模块的对外接口(js文件中哪些内容对外公开|导出)。

import 输入其它模块提供的功能。

项目中公共方法

定义一个common.js文件，文件中定义项目的公共方法

```
//显示当前时间 yyyy-MM-dd
//2025-06-01
export function getCurDate(){
    let d = new Date();
    let year = d.getFullYear(); //得到年份
    let month = d.getMonth()+1; //枚举方式
    let day = d.getDate();

    month = month<10?'0'+month:month;
    day = day<10?'0'+day:day;

    return `${year}-${month}-${day}`;
}

//SessionStorage存 key-value
export function setSessionStorage(key,value){
    //传入value可以是任何类型，传入value可以json对象。
    //sessionStorage 存储只能是存字符串数据
```

```

    sessionStorage.setItem(key, JSON.stringify(value));
  }

  //获取SessionStorage的数据
  export function getSessionStorage(key){
    let str = sessionStorage.getItem(key);
    if(str===''||str===null||str==='null'||str===undefined){
      return null;
    }else{
      return JSON.parse(str); //将字符串转换json对象
    }
  }

  //删除SessionStorage的方法
  export function removeSessionStorage(key){
    sessionStorage.removeItem(key);
  }

  //localStorage存 key-value
  export function setLocalStorage(key,value){
    localStorage.setItem(key,JSON.stringify(value));
  }

  //获取localStorage的数据
  export function getLocalStorage(key){
    let str = localStorage.getItem(key);
    if(str===''||str===null||str==='null'||str===undefined){
      return null;
    }else{
      return JSON.parse(str); //将字符串转换json对象
    }
  }

  //删除localStorage的方法
  export function removeLocalStorage(key){
    localStorage.removeItem(key);
  }
}

```

公共组件 Footer.vue显示

需要改变App.vue主组件的样式

```

<template>
  <!-- 路由出口 -->
  <router-view/>
</template>

<style>
  /* 主组件：所有组件都使用的样式。 */
  /* CSS重置 */
  body,div,u1,o1,li,p,h1,h2,h3,h4,h5,h6,span,a,img,form{
    margin: 0;
    padding: 0;
  }
  html,body{
    width: 100%;
    font-family: Arial,'Microsoft Yahei';
  }

```



```

    height:100%;
  }
  ul{
    list-style: none; /* none 设置列表不显示列表符号 */
  }
  a{ text-decoration: none; } /* 去掉页面上所有超链接下划线 */
</style>

```

```

<script>
export default {
  data() {
    return {
      //组件页面所需要的所有的变量 属性:值,
      name: '上海海洋',
      addr: '上海市临港',
    };
  },
  methods: {
    //组件页面所需的所有方法 定义忽略function关键词
    test(){

    },
    test2(){

    }
  },
};
</script>

```

Footer.vue编码

```

<template>
  <ul class="footer">
    <li @click="toHome()">
      <i class="icon icon-home" />
      <p>首页</p>
    </li>
    <li @click="toCart()">
      <i class="icon icon-cart" />
      <p>购物车</p>
    </li>
    <li @click="toOrder()">
      <i class="icon icon-order"/>
      <p>订单</p>
    </li>
    <li @click="toCenter()">
      <i class="icon icon-me" />
      <p>我的</p>
    </li>
  </ul>
</template>

<script>
export default {
  data() {
    return {
    };
  },
},

```

```

methods: {
  toHome(){
    // this.$router获得路由对象，调用push() 跳转至指定路径
    // 重复路由报错，处理异常，加上 .catch(e=>{})
    this.$router.push('/').catch(e=>{})
  },
  toCart(){
    this.$router.push('/cart').catch(e=>{})
  },
  toOrder(){
    this.$router.push('/orderList').catch(e=>{})
  },
  toCenter(){
    this.$router.push('/center').catch(e=>{});
  }
},
};
</script>

<style scoped>
.icon{ width:8vw; height: 8vw; display: block; background-size: cover; }
.icon-me{ background-image: url(../assets/me.png); }
.icon-order{ background-image: url(../assets/order.png); }
.icon-home{ background-image: url(../assets/home.png); }
.icon-cart{ background-image: url(../assets/cart.png); }

/*/////底部导航栏/////*/
.wrapper .footer{
  width: 100%;
  height: 14vw;
  border-top: solid 1px #ddd;
  background-color: #fff;
  position: fixed;
  left: 0;
  bottom: -0.1vw;
  display: flex;
  justify-content: space-around;
  align-items: center;
}
.wrapper .footer li{
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  color: #79859E;
  user-select: none;
  cursor: pointer;
}
.wrapper .footer li p{ font-size:2.8vw; }
.wrapper .footer li i{ font-size: 5vw; }
</style>

```

Vue3语法

组合化

```

<script>
  export default{
    setup(){
      //代码
    }
  }
</script>
变为
<script setup>
  //代码
</script>

```

响应式变量

- reactive() 定义 如果定义数据是一个对象，建议使用reactive

```

import {reactive,ref} from "vue"

const obj = reactive({
  count:200,
})

const add=()=>{
  obj.count++;
}

//举例，变量数据就是一个Student对象
const student = reactive({
  name: '站三',
  age:22,
  phone: '19990099090'
})

```

- ref()定义 如果定义变量就是一个普通数据，是一个单一数据

```

import {reactive,ref} from "vue"

//代码 变量(普通，响应式)、方法
const count = ref(100);
//定义为空数组
const arr = ref([]);

//++count方法
const add = ()=>{
  count.value ++;
}

//举例 int a=100;
const a = ref(100);

```

Vue3语法修改 Footer.vue、Home.vue

```

<script setup>
import {useRouter} from "vue-router"

//创建路由对象，等价于vue2语法书写的this.$router
const router = useRouter();

```

```
const toHome = ()=>{
  router.push('/');
}

const toUserCenter = ()=>{
  router.push('/login')
}
</script>
```

Home.vue

截图

实现登录

安装 ElementUI Plus插件库

启动vsCode终端，使用npm工具安装

```
npm install element-plus --save
```

需要再main.js中设置插件

```
import ElementPlus from 'element-plus';
import '/node_modules/element-plus/dist/index.css';

app.use(ElementPlus);
```

编写登录页面

```
<template>
  <div class="wrapper">
    <header>
      <p>用 户 登 录</p>
    </header>

    <!-- 登录表单 ref="loginForm"相当于定义 id="xxx" 在js代码中可以使用loginForm名字获得该
    Form表单对象
      v-model 双向数据绑定，变量中值可以显示在 文本框，
      文本框的值修改，也可以同步回变量
    -->
    <el-form ref="loginForm" :model="account" label-width="80px" class="form-box"
    :rules="rules">
      <el-form-item label="账户" prop="accountId">
        <el-input v-model="account.accountId" placeholder="请输入登录账户" />
      </el-form-item>
      <el-form-item label="密码" prop="password">
        <el-input v-model="account.password" type="password" show-password
placeholder="请输入密码" />
      </el-form-item>

      <div class="button-login">
        <button @click="login()">登录</button>
      </div>
      <div class="button-register">
```

```

        <button @click="toRegister()">去注册</button>
      </div>
    </el-form>
    <Footer></Footer>
  </div>
</template>

<script setup>
import Footer from '@/components/Footer.vue';
import {ref, reactive} from "vue"
import {useRouter} from "vue-router"

//ElementUI-Plus验证规则
const rules = reactive({
  accountId:[
    { required: true, message: '请输入登录账户', trigger: 'blur' }
  ],
  password:[
    { required: true, message: '请输入密码', trigger: 'blur' }
  ],
})

const router = useRouter();

//定义变量 登录用户
const account = reactive({
  accountId:'', //登录手机号
  password:'', //密码
})

//获得登录表单对象
const loginForm = ref(null);

const login = () =>{
  //通过loginForm表单对象，调用该对象 validate()验证方法，如果定义验证规则通过，v是true，否则是false
  loginForm.value.validate((v)=>{
    if(v){
      alert('请求服务器进行登录');
    }else{
      return false;
    }
  });
}

const toRegister = ()=>{
  router.push('/register');
}

</script>

<style scoped>
.wrapper header{
width:100%; height:12vw;
background: linear-gradient(to right,#fff1eb,#ace0f9);
color:#596164;
font-size: 5vw;
position: fixed; left:0; top: 0;
display: flex;
justify-content: center;
align-items: center;

```

```
    z-index: 1000;
}

.wrapper header p{
    letter-spacing: 2vw;
}
/* APP标题部分样式 */

/* 登录部分样式代码 */
.wrapper .form-box{
    width:100%; margin-top:20vw;
    padding: 5vw;
    box-sizing: border-box;
}

.wrapper .form-box .el-input{
    width:90%;
}

.wrapper .button-login{
    width:100%;
    box-sizing: border-box;
    padding:4vw 4vw 0 4vw;
}

.wrapper .button-login button{
    width:100%;
    height:10vw;
    font-size: 3.8vw;
    background-color: #3cba92;
    border:none;
    border-radius: 1.5vw;
    color:#efefef;
    outline: none;
    letter-spacing: 2vw;
}

.wrapper .button-register{
    width:100%;
    box-sizing: border-box;
    padding:4vw;
}

.wrapper .button-register button{
    width:100%;
    height:10vw;
    font-size: 3.8vw;
    background-color: #fdfcfb;
    border:0.3vw solid #e2d1c3;
    border-radius: 1.5vw;
    color:#5f72bd;
    outline: none;
    letter-spacing: 2vw;
}
</style>
```

