VUE培训及说明文档

2023年8月30日

### 目录：

* 一、前言
* 二、Vue概述
* 三、MVVM架构
* 四、第一个Vue程序
* 五、Vue七大属性
* 六、常用指令
* 七、数据双向绑定
* 八、Vue组件
* 九、Vue生命周期
* 十、Axios异步通信
* 十一、Vue计算属性
* 十二、插槽slot
* 十三、自定义事件内容分发
* 十四、vue-cli脚手架
* 十五、Webpack打包器
* 十六、vue-router路由
* 十七、vue + elementUI整合小案例
* 十八、路由嵌套
* 十九、参数传递
* 二十、路由模式与处理404
* 二十一、路由钩子与异步请求

**一、前言**

Vue 是一套用于构建用户界面的渐进式框架 。

与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。

最初它不过是个人项目，时至今日，已成为全世界三大前端框架之一，github 上拥有 17.8万 Star。 领先于 React 和 Angular，在国内更是首选。



Vue发展史：

2013年，在 Google 工作的尤雨溪，受到 Angular 的启发，开发出了一款轻量级框架，最初命名为 Seed 。

2013年12月，更名为 Vue，图标颜色是代表勃勃生机的绿色，版本号是 0.6.0。

2014.01.24，Vue 正式对外发布，版本号是 0.8.0。

2014.02.25，0.9.0 发布，有了自己的代号：Animatrix，此后，重要的版本都会有自己的代号。

2015.06.13，0.12.0，代号Dragon Ball，Laravel 社区（一款流行的 PHP 框架的社区）首次使用 Vue，Vue 在 JS 社区也打响了知名度。

2015.10.26，1.0.0 Evangelion 是 Vue 历史上的第一个里程碑。同年，vue-router、vuex、vue-cli 相继发布，标志着 Vue从一个视图层库发展为一个渐进式框架。

2016.10.01，2.0.0 是第二个重要的里程碑，它吸收了 React 的虚拟 Dom 方案，还支持服务端渲染。自从Vue 2.0 发布之后，Vue 就成了前端领域的热门话题。

2019.02.05，Vue 发布了 2.6.0 ，这是一个承前启后的版本，在它之后，推出了 3.0.0。

2019.12.05，在万众期待中，尤雨溪公布了 Vue 3 源代码，此时的 Vue 3仍 处于 Alpha 版本。

2020年09月18日，Vue.js 3.0 正式发布。

**二、Vue概述**

Vue是一套用于构建用户界面的**渐进式框架**，该框架可以**自底向上**逐层应用。**Vue的核心库只关注视图层**，不仅简单而且便于与**第三方库**(如: vue-router: 跳转，vue-resource: 通信，vuex:管理)或项目整合

Vue是渐进式JavaScript框架，所谓**渐进式**就是逐步实现新特性的意思，如实现**模块化开发、路由、状态管理**等新特性。其特点是综合了**Angular（模块化）** 和 **React（虚拟DOM）** 的优点，并有自己的特色功能：**计算属性**

**三、MVVM架构**

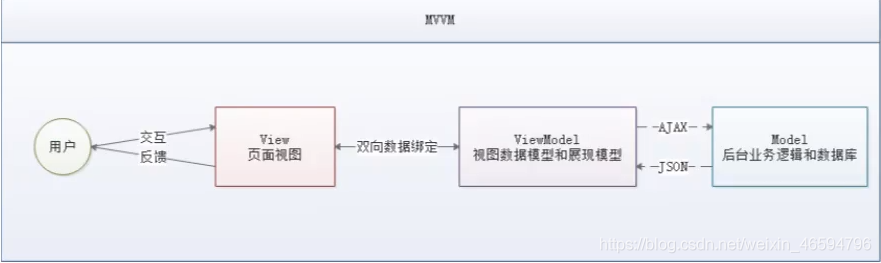
1.什么是MVVM

**MVVM（Model-View-ViewModel）** 源自于经典的MVC (Model-View-Controller) 模式。MVVM的核心是**ViewModel**层，负责转换Model中的数据对象来让数据变得更容易管理和使用，其作用如下:

* 该层向上与视图层进行双向数据绑定
* 向下与Model层通过接口请求进行数据交互

ViewModel能够观察到**数据**变化，并对视图对应的内容进行更新

ViewModel能够监听到**视图**变化，并能够通知数据发生变化

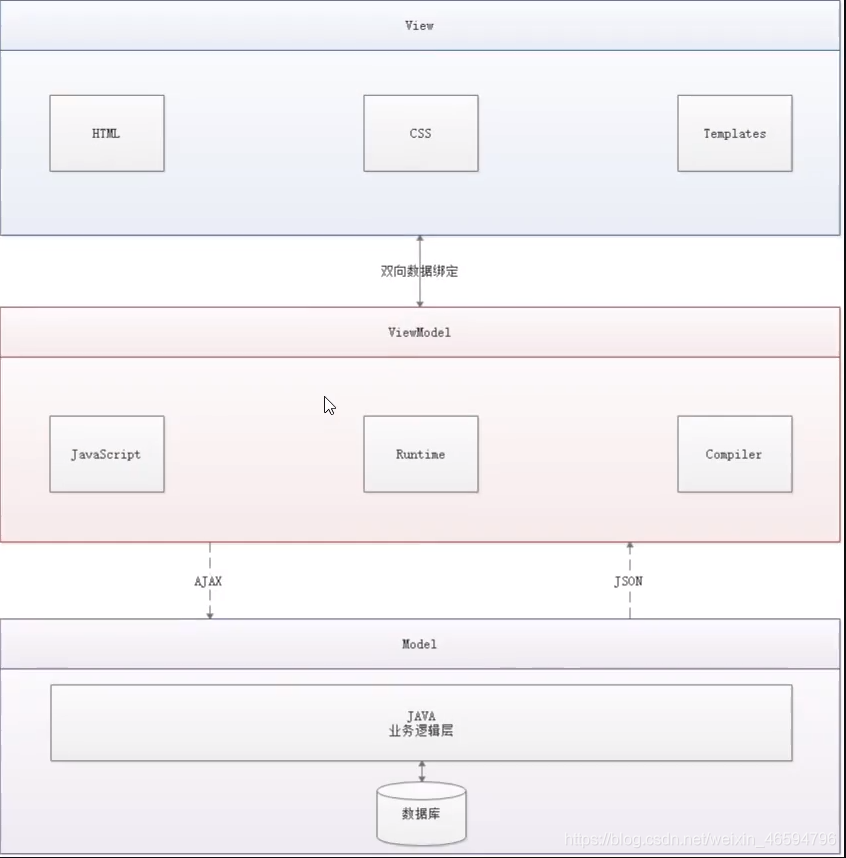


2.MVVM的好处

MVVM模式和MVC模式一样，主要目的是分离视图(View)和模型(Model),有几大好处：

* 低耦合：视图(View)可以独立于Model变化和修改,一个ViewModel可以绑定到不同的View上，当View变化的时候Model可以不变，当Model变化的时候View也可以不变。
* 可复用：你可以把一些视图逻辑放在一个ViewModel里面，让很多View重用这段视图逻辑。
* 独立开发：开发人员可以专注于业务逻辑和数据的开发(ViewModel),设计人员可以专注于页面设计。
* 可测试：界面素来是比较难于测试的，而现在测试可以针对ViewModel来写。

3.MVVM图解

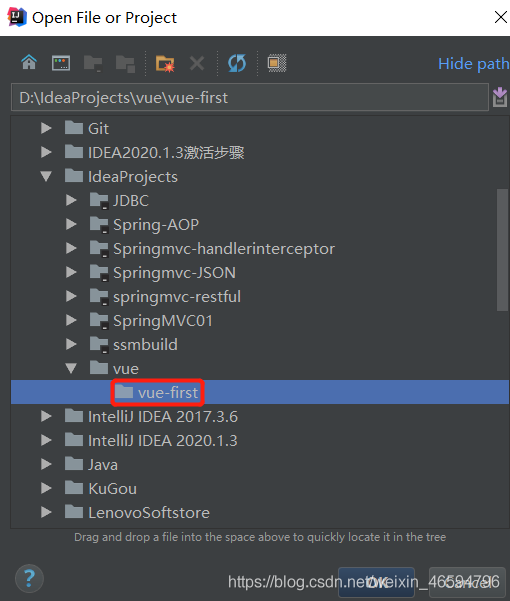


* Model : 模型层，在这里表示JavaScript对象
* View : 视图层,在这里表示DOM (HTML操作的元素)
* ViewModel : 连接视图和数据的中间件，Vue.js就是MVVM中的ViewModel层的实现者在MVVM架构中，是不允许数据和视图直接通信的，只能通过ViewModel来通信，而ViewModel就是定义了一个Observer观察者
* ViewModel来通信，而ViewModel就是定义了一个Observer观察者
* ViewModel 能够观察到数据的变化，并对视图对应的内容进行更新
* ViewModel 能够监听到视图的变化，并能够通知数据发生改变

至此我们就明白了，Vue就是一个MVVM的实现者，它的核心就是实现了**DOM监听**与**数据绑定**

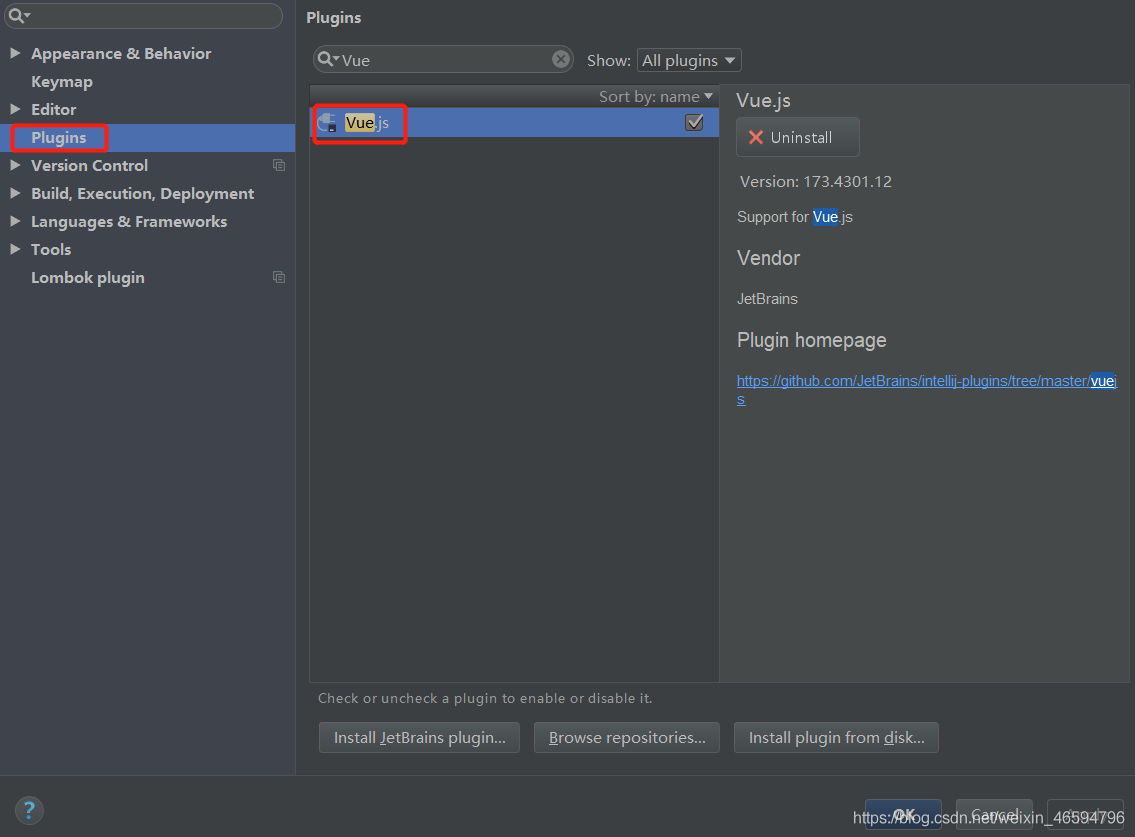
**四、第一个Vue程序**

我们在硬盘中创建一个文件夹，叫做vue-first，我们使用IDEA通过Open的方式打开vue-first这个空文件夹：



我们在创建的项目中，创建一个index.html文件，就可以开始我们第一个Vue程序的编写了！

但是在编写代码之前，我们需要打开setting找到Plugins，安装好Vue的插件：



第一个Vue程序代码展示：

<!DOCTYPE html>

<html lang="en">

<head><meta charset="UTF-8"><title>Title</title><!--1.导入vue.js--><script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.min.js"></script>

</head>

<body><!--View层 模板-->

<div id="app">{{message}}

</div><script>var vm = new Vue({el : '#app',/\*Model层 数据\*/data : {message : "Hello Vue"}});

</script></body>

</html>



**五、Vue七大属性**

1. el：用来指示vue编译器从什么地方开始解析vue语法
2. data：用来组织从view中抽象出来的属性，可以说将视图的数据抽象出来存放在data中
3. template：用来设置模板，会替换页面元素
4. methods：放置在页面中的业务逻辑，方法一般放在methods中
5. render：创建真正的虚拟DOM
6. computed：用于计算属性
7. watch：监听data中的数据变化，两个参数，一个返回新值，一个返回旧值

watch: function(new,old){}

**六、常用指令**

我们接下来来讲解指令，指令的前缀有v-，它们会在渲染的DOM上应用特殊的响应式行为。

1.绑定属性（v-bind）

我们还可以使用v-bind来绑定元素特性

<!DOCTYPE html>

<html lang="en" xmlns:v-bind="http://www.w3.org/1999/xhtml">

<head><meta charset="UTF-8"><title>Title</title><!--1.导入vue.js--><script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.min.js"></script>

</head>

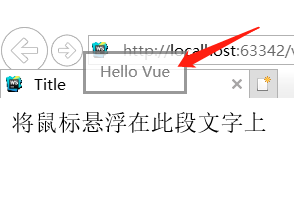
<body><!--View层 模板-->

<div id="app"><span v-bind:title="message">将鼠标悬浮在此段文字上</span>

</div><script>var vm = new Vue({el : '#app',/\*Model层 数据\*/data : {message : "Hello Vue"}});

</script></body>

</html>



2.判断（v-if）

<!DOCTYPE html>

<html lang="en">

<head><meta charset="UTF-8"><title>Title</title><!--1.导入vue.js--><script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.min.js"></script>

</head>

<body>

<!--View层-->

<div id="app"><h1 v-if="ok">Yes</h1><h1 v-else>No</h1>

</div><script>var app = new Vue({el: '#app',data: {ok: true}})

</script>

</body>

</html>



3.循环（v-for）

<!DOCTYPE html>

<html lang="en">

<head><meta charset="UTF-8"><title>Title</title><!--1.导入vue.js--><script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.min.js"></script>

</head>

<body>

<!--View层-->

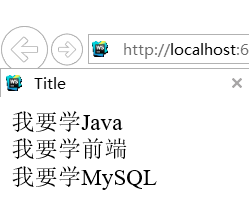
<div id="app"><li v-for="item in items">{{item.message}}</li>

</div><script>var ***app*** = new Vue({el: '#app',data: {items: [{message: '我要学Java'},{message: '我要学前端'},{message: '我要学MySQL'}]}})

</script>

</body>

</html>



4.绑定事件（v-on）

可以用v-on指令监听DOM事件，并在触发时运行一些JavaScript代码

<!DOCTYPE html>

<html lang="en" xmlns:v-on="http://www.w3.org/1999/xhtml">

<head><meta charset="UTF-8"><title>Title</title><!--1.导入vue.js--><script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.min.js"></script>

</head>

<body>

<!--View层-->

<div id="app"><button v-on:click="sayHi">click me!</button>

</div><script>var ***app*** = new Vue({el: '#app',data: {message: "我爱敲代码"},//方法必须定义在Vue的methods对象中，通过v-on绑定

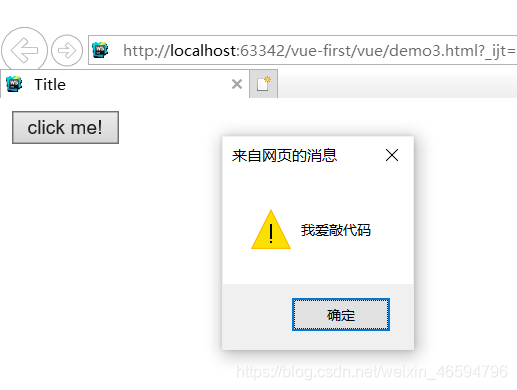
methods: {

sayHi: function () {alert(this.message)}}})

</script>

</body>

</html>



**总结**：我们想要数据放在data属性中，想要方法放在methods属性中。想要绑定属性用v-bind，想要绑定事件用v-on，想要绑定数据用{{ }}

**七、数据双向绑定**

双向绑定是vue的精髓，当数据发生变化时，视图就发生变化。当视图发生变化时，数据也跟着发生变化！

<!DOCTYPE html>

<html lang="en" xmlns:v-on="http://www.w3.org/1999/xhtml">

<head><meta charset="UTF-8"><title>Title</title><!--1.导入vue.js--><script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.min.js"></script>

</head>

<body>

<!--View层-->

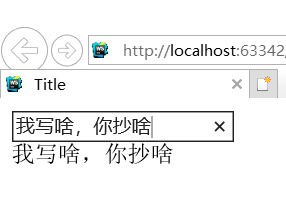
<div id="app"><input type="text" v-model="message"><br>{{message}}

</div><script>var ***app*** = new Vue({el: '#app',data: {message: "我写啥，你抄啥"}})

</script>

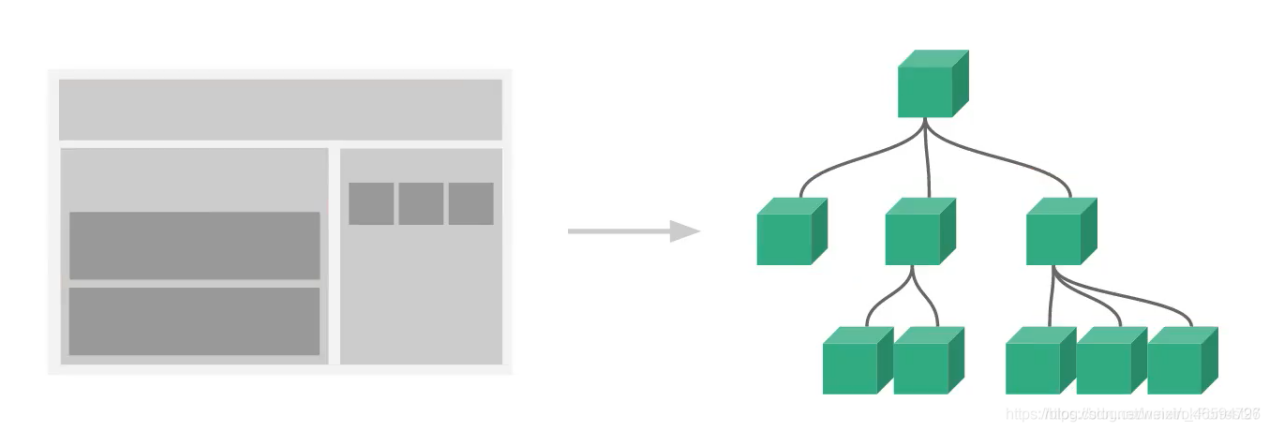
</body>

</html>



**八、Vue组件**

**组件是可复用的Vue实例**，说白了就是一组可以重复使用的模板，跟JSTL的自定义标签、Thymeleaf的th:fragment 等框架有着异曲同工之妙。通常一个应用会以一棵嵌套的组件树的形式来组织：



如果我们需要传递参数到组件，此时就可以使用props属性

<!DOCTYPE html>

<html lang="en" xmlns:v-on="http://www.w3.org/1999/xhtml">

<head><meta charset="UTF-8"><title>Title</title><!--1.导入vue.js--><script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.min.js"></script>

</head>

<body>

<!--View层-->

<div id="app"><xu v-for="item in items" v-bind:bin="item"></xu>

</div><script>//Vue.component()方法注册一个组件

//

Vue.component("xu",{

//传递参数用props接收

props: ['bin'],//组件模板，可以使用props中的参数

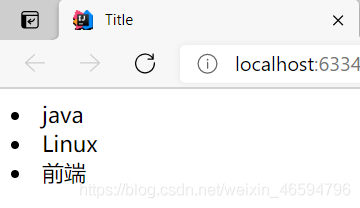
template: '<li>{{bin}}</li>'})

var ***app*** = new Vue({el: '#app',data: {items: ["java","Linux","前端"]}})

</script>

</body>

</html>



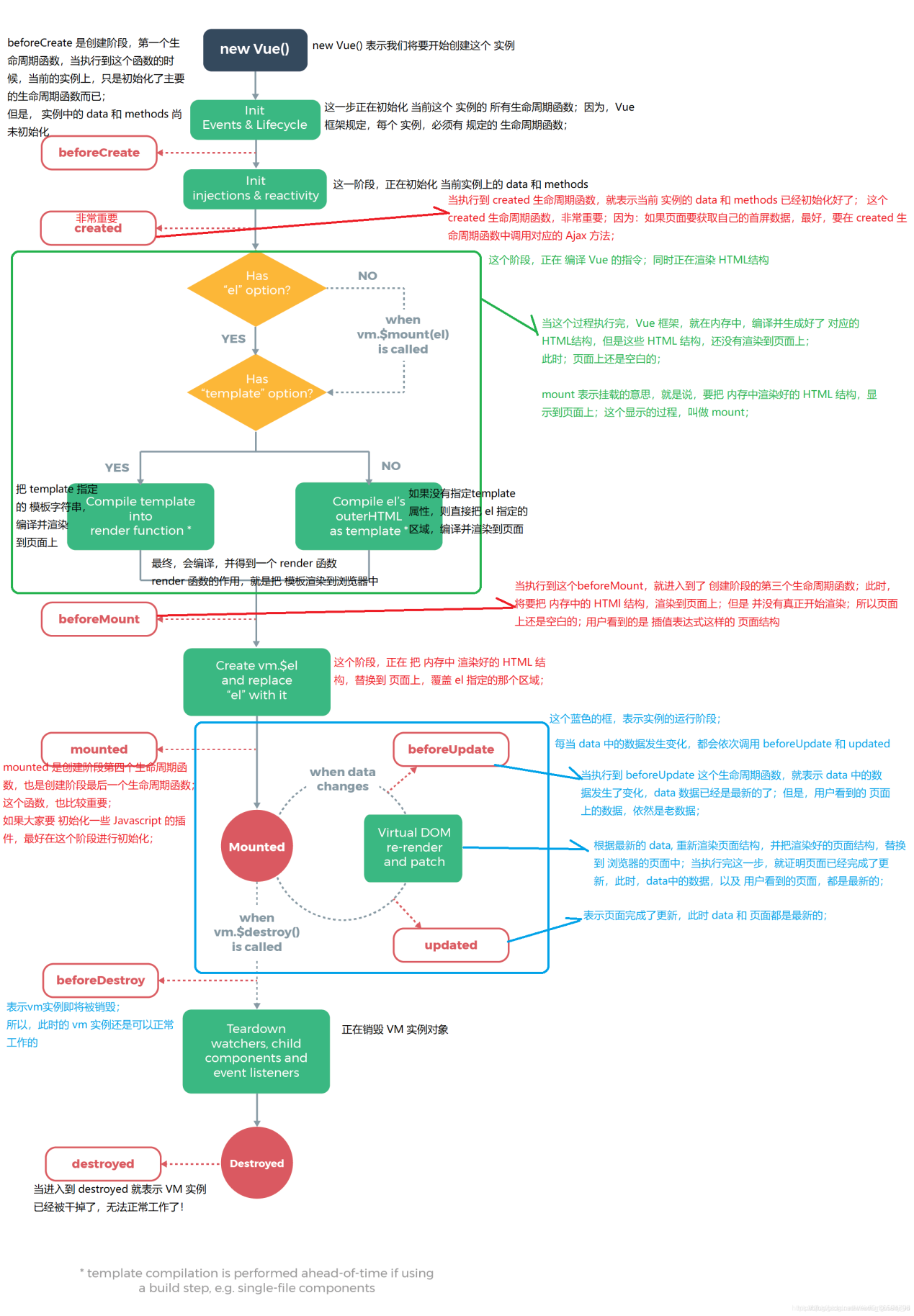
**九、Vue生命周期**

1.Vue生命周期理解

Vue实例有一个完整的生命周期，也就是从开始创建、初始化数据、编译模板、挂载DOM、渲染→更新→渲染、卸载等一系列过程，我们称这是Vue的生命周期。通俗说就是Vue实例从创建到销毁的过程，就是生命周期。

在Vue的整个生命周期中，它提供了一系列的事件，可以让我们在事件触发时注册JS方法,可以让我们用自己注册的JS方法控制整个大局，在这些事件响应方法中的this直接指向的是Vue的实例。

2.Vue生命周期图解



**十、Axios异步通信**

由于Vue.js是一个视图层框架且作者(尤雨溪) 严格准守SoC (关注度分离原则)，所以Vue.js并**不包含AJAX**的通信功能，为了解决通信问题，作者单独开发了一个名为vue-resource的插件，不过在进入2.0 版本以后停止了对该插件的维护并推荐了**Axios** 框架。少用jQuery，因为它操作Dom太频繁!

先在IDEA中创建一个data.json文件，模拟Json数据：

{

"name":"MoYu",

"url":"https://www.moyuzc.cn/",

"page":1,

"isNonProfit":true,

"address": {

"street":"含光门",

"city":"陕西西安",

"country":"中国"

},

"links":[

{

"name": "bilibili",

"url": "https://space.bilibili.com/84493733"

},

{

"name": "MoYu",

"url": "https://moyu-zc.gitee.io/"

}

]

}

异步通信代码展示：

<div id="vue">

<div>

{{info.name}}

<br>

{{info.address}}

</div>

</div>

<script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js"></script>

<script src="https://unpkg.com/axios/dist/axios.min.js"></script>

<script type="text/javascript">

var ***vm*** = new Vue({

el:'#vue',

data(){

return{

//请求的返回参数，必须和 json 字符串一样

info:{

name: null,

address: {

street: null,

city: null,

country: null

},

}

}

},

mounted(){ //钩子函数 链式编程 ES6新特性

axios.get('data.json').then(response=>(***console***.log(this.info=response.data)));

}

});

</script>



**十一、Vue计算属性**

计算属性的重点突出在属性两个字上，首先它是个属性，其次这个属性有计算的能力，这里的计算就是个函数。

简单点说，它就是一个能够将计算结果缓存起来的属性(将行为转化成了静态的属性)，可以想象为**缓存**

<!DOCTYPE html>

<html lang="en" xmlns:v-bind="http://www.w3.org/1999/xhtml">

<head><meta charset="UTF-8"><title>Title</title><!--导入vue.js--><script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.min.js"></script>

</head>

<body>

<div id="vue"><p>{{currentTime1()}}</p><p>{{currentTime2}}</p>

</div><script>var ***app*** = new Vue({el: '#vue',methods: {currentTime1: function () {return ***Date***.now()}},//计算属性：methods和computed中的方法不能重名，重名了只会调用methods的方法

computed: {currentTime2: function () {return ***Date***.now()}}})

</script>

</body>

</html>

**说明**：

* methods：定义方法，调用方法需要带括号
* computed：定义计算属性，因为是属性所以调用不需要带括号

**结论**：调用方法时，每次都需要进行计算，既然有计算过程则必定产生系统开销，那如果这个结果是不经常变化的呢?此时就可以考虑将这个结果缓存起来，采用计算属性可以很方便的做到这一点，**计算属性的主要特性就是为了将不经常变化的计算结果进行缓存，以节约我们的系统开销**

**十二、插槽slot**

在Vue.js中我们使用 元素作为承载分发内容的出口，作者称其为插槽，可以应用在组合组件的场景中;

<!DOCTYPE html>

<html lang="en" xmlns:v-bind="http://www.w3.org/1999/xhtml">

<head><meta charset="UTF-8"><title>Title</title><!--导入vue.js--><script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.min.js"></script>

</head>

<body>

<div id="app"><todo><todo-title slot="todo-title" v-bind:title="title"></todo-title><todo-items slot="todo-items" v-for="item in todoItems" v-bind:item="item"></todo-items></todo>

</div><script>//slot 插槽 这个组件要定义在前面不然出不来数据

Vue.component("todo", {template: '<div>\<slot name="todo-title"></slot>\<ul>\<slot name="todo-items"></slot>\</ul>\<div>'});

Vue.component("todo-title",{props: ['title'],template: '<div>{{title}}</div>'});

Vue.component("todo-items",{props: ['item'],template: '<li>{{item}}</li>'})

var ***app*** = new Vue({el: '#app',data: {title: "学习内容",todoItems: ['学习Java','学习前端','学习运维']}})

</script>

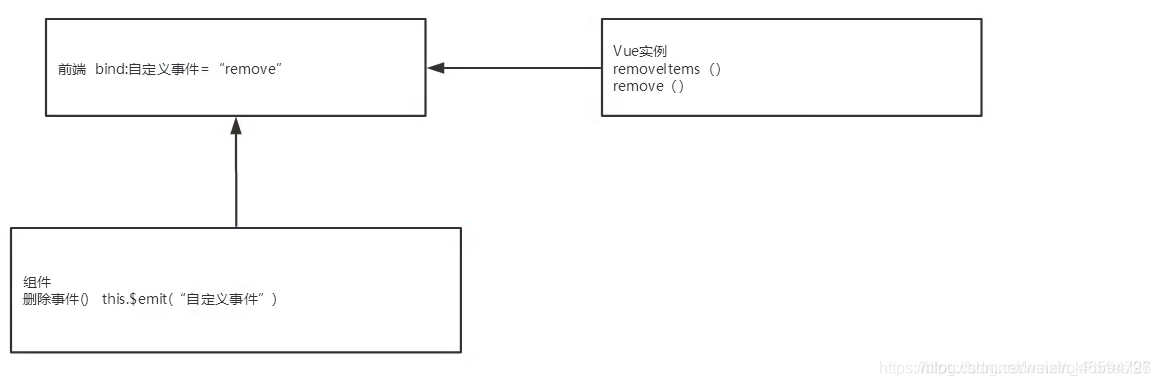
</body>

</html>



**十三、自定义事件内容分发**

通过以上代码不难发现，数据项在Vue的实例中，但删除操作要在组件中完成，那么组件如何才能删除Vue实例中的数据呢?此时就涉及到参数传递与事件分发了，Vue为我们提供了自定义事件的功能很好的帮助我们解决了这个问题;**使用this.$emit(‘自定义事件名’,参数)**



<!DOCTYPE html>

<html lang="en" xmlns:v-bind="http://www.w3.org/1999/xhtml">

<head><meta charset="UTF-8"><title>Title</title><!--导入vue.js-->

<script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.min.js"></script>

</head>

<body>

<div id="app"><todo><todo-title slot="todo-title" v-bind:title="title"></todo-title><todo-items slot="todo-items" v-for="(item,index) in todoItems" v-bind:item="item" v-bind:index="index"v-on:remove="removeItems(index)" v-bind:key="index"></todo-items></todo>

</div>

<script>

//slot 插槽 这个组件要定义在前面不然出不来数据

Vue.component("todo", {template: '<div>\<slot name="todo-title"></slot>\<ul>\<slot name="todo-items"></slot>\</ul>\<div>'});

Vue.component("todo-title",{props: ['title'],template: '<div>{{title}}</div>'});

Vue.component("todo-items",{props: ['item','index'],template: '<li>{{item}}<button v-on:click="remove">删除</button></li>',methods: {remove: function (index) {

//this.$emit 自定义事件分发

this.$emit('remove',index)}}})

var ***app*** = new Vue({el: '#app',data: {title: "学习内容",todoItems: ['学习Java','学习前端','学习运维']},methods: {removeItems:

function (index) {this.todoItems.splice(index,1); //一次删除一个元素

}

}

})

</script>

</body>

</html>

**十四、vue-cli脚手架**

1.什么是vue-cli

vue-cli 官方提供的一个脚手架,用于快速生成一个 vue 的项目模板;

预先定义好的目录结构及基础代码，就好比咱们在创建 Maven 项目时可以选择创建一个骨架项目，这个骨架项目就是脚手架,我们的开发更加的快速;

主要目的：

* 统一目录结构
* 本地调试
* 热部署
* 单元测试
* 集成打包上线

2.使用vue-cli的环境前提

我们使用vue-cli之前，需要安装node.js！

安装网址：http://nodejs.cn/download/

安装成功后：

* cmd中输入node -v，查看是否能正确打印版本号
* cmd中输入npm -v，查看是否能正确打印版本号

安装完node.js之后，就自带npm，这个**npm就是一个软件包管理工具（类似Maven）**！

为了让我们node.js运行更快，我们需要安装node.js的**淘宝镜像加速器（cnpm）**

# -g 就是全局安装

npm install cnpm -g

# 或使用如下语句解决 npm 速度慢的问题

通过输入“npm install -g cnpm --registry=http://registry.npmmirror.com“来[安装cnpm](https://so.csdn.net/so/search?q=%E5%AE%89%E8%A3%85cnpm&spm=1001.2101.3001.7020" \t "_blank)。安装成功后，使用cnpm代替npm来安装依赖包。

 通过命令“npm config set registry http://registry.npmmirror.com“设置淘宝镜像源，使用[npm安装](https://so.csdn.net/so/search?q=npm%E5%AE%89%E8%A3%85&spm=1001.2101.3001.7020" \t "_blank)包时，会自动使用淘宝的镜像源

安装完淘宝镜像加速器之后，安装**vue-cli**

#在命令台输入

cnpm install vue-cli -g

#查看是否安装成功

vue list

3.创建一个vue-cli程序

创建一个文件夹，作为vue-cli程序的文件夹：D:\IdeaProjects\vue

创建一个基于webpack模板的vue应用程序：

# 这里的myvue是项目名称

vue init webpack myvue

项目下载完成后，cmd中的选项一路选No

然后我们需要在命令行中输入下列三个命令：

//进入myvue项目中

cd myvue//安装该项目的所有依赖环境

npm install//启动了当前项目

npm run dev

这样我们就创建完了第一个vue-cli项目

**十五、Webpack打包器**

1.什么是webpack

webpack是一个现代JavaScript应用程序的**静态模块打包器(module bundler)**。当webpack处理应用程序时，它会递归地构建一个依赖关系图(dependency graph),其中包含应用程序需要的每个模块，然后将所有这些模块打包成一个或多个bundle.

Webpack是当下最热门的**前端资源模块化管理和打包工具**，它可以将许多松散耦合的模块按照依赖和规则打包成符合生产环境部署的前端资源。还可以将按需加载的模块进行代码分离，等到实际需要时再异步加载。

2.安装webpack

webpack是一个模块加载器兼打包工具，它能把各种资源都作为模块来处理和使用，使用该工具的**目的**：把ES6规范的代码打包编译成ES5规范的代码，使所有浏览器都能运行！

安装命令：

npm install webpack -g

npm install webpack-cl -g

测试成功安装：

webpack -v

webpack-cli -v

这样我们就安装完成了！

3.使用webpack

创建一个文件夹：D:\IdeaProjects\vue\webpack-study

使用idea打开该文件夹！

在idea中创建一个modules目录，里面创建一个hello.js文件：

//暴露一个方法

exports.sayHi = function () {document.write("<h1>徐某学ES6</h1>")

}

创建main.js 当作是js主入口 , main.js 请求hello.js 调用sayHi()方法：

var hello = require("./hello")

hello.sayHi();

在主目录创建webpack.config.js , webpack.config.js 这个相当于webpack的配置文件：

module.exports = {//enrty请求main.js的文件entry: './modules/main.js',//output是输出的位置和名字output: {filename: "./js/bundle.js"}

};

在idea命令台输入webpack命令（idea要设置管理员启动）

完成上述操作之后会在主目录生成一个dist文件 生成的js文件夹路径为 **/dist/js/bundle.js**

接下来，在主目录创建index.html 导入bundle.js

<!DOCTYPE html>

<html lang="en">

<head><meta charset="UTF-8"><title>Title</title>

</head>

<body><!--前端的模块化开发-->

<script src="dist/js/bundle.js"></script></body>

</html>



**十六、vue-router路由**

Vue Router是Vue.js官方的路由管理器（**视图跳转**）。它和Vue.js的核心深度集成，让构建单页面应用变得易如反掌

vue-router 是一个插件包，所以我们还是需要用 npm/cnpm 来进行安装的。打开命令行工具，进入你的项目目录，输入下面命令：

npm install vue-router --save-dev

在components目录下存放我们编写的组件，定义Content.vue和Main.vue两个组件：

<template><h1>内容页</h1>

</template><script>export default {name: "Content"}

</script><style scoped></style>========================================================================<template><h1>首页</h1>

</template><script>export default {name: "Main"}

</script><style scoped></style>

创建一个router目录，专门存放路由，文件名为index.js：

import Vue from 'vue'

//导入路由插件

import VueRouter from 'vue-router'

//导入两个定义的组件

import Content from '../components/Content'

import Main from '../components/Main'//安装路由

Vue.use(VueRouter)//配置导出路由

export default new VueRouter({routes: [{//路由路径path: '/content',//跳转的组件component: Content},{//路由路径path: '/main',//跳转的组件component: Main}]

});

在main.js中配置路由：

import Vue from 'vue'

import App from './App'

import router from './router' //自动扫描里面的路由配置new Vue({el: '#app',//配置路由router,components: { App },template: '<App/>'

})

在App.vue中使用路由：

<template><div id="app"><h1>Vue-Router</h1><router-link to="/main">首页</router-link><router-link to="/content">内容页</router-link><!--展示template模板--><router-view></router-view></div>

</template><script>export default {name: 'App'

}

</script><style>

#app {font-family: 'Avenir', Helvetica, Arial, sans-serif;-webkit-font-smoothing: antialiased;-moz-osx-font-smoothing: grayscale;text-align: center;color: #2c3e50;margin-top: 60px;

}

</style>

启动测试一下 ： npm run dev

其实，main.js配置完了就基本不需要变动了，我们只需要添加组件，在index.js中配置组件路径，最后在App.vue中添加router-link，基本就足够了！

三者关系：**index.html 调用 main.js 调用 App.vue**

**十七、vue + elementUI整合小案例**

在Idea项目目录中创建一个hello-vue项目，在该目录下可以使用命令行（命令行要使用**管理员**模式！）：

//创建一个基于webpack模板的vue应用程序

vue init webpack hello-vue

安装依赖，我们需要安装vue-router，element-ui，sass-loader和node-sass四个插件：

//进入目录

cd hello-vue

//安装vue-router

npm install vue-router@3.5.2 --save-dev

//安装element ui

npm i element-ui -S npm install element-plus --save

//安装依赖

npm install

//安装 SASS 加载器

cnpm install sass-loader node-sass --save-dev

//启动测试

npm run dev

npm命令解释：

安装模块到项目目录下

npm install moduleName-g 的意思是将模块安装到全局，具体安装到磁盘的哪个位置，要看 npm config prefix的位置

npm install -g moduleName–save的意思是将模块安装到项目目录下，并在package文件的dependencies节点写入依赖，-S为该命令的缩写

npm install moduleName -save–save-dev的意思是将模块安装到项目目录下，并在package文件的devDependencies节点写入依赖，-D为该命令的缩写

npm install moduleName -save-dev

接下来，我们打开IDEA将我们创建的项目open一下！

创建一个views目录，编写两个组件：Login.vue和Main.vue

<template><h1>首页</h1>

</template><script>export default {name: "Main"}

</script><style scoped></style>========================================================================<template xmlns:v-on=""><div><el-form ref="loginForm" :model="form" :rules="rules" label-width="80px" class="login-box"><h3 class="login-title">欢迎登录</h3><el-form-item label="账号" prop="username"><el-input type="text" placeholder="请输入账号" v-model="form.username"/></el-form-item><el-form-item label="密码" prop="password"><el-input type="password" placeholder="请输入密码" v-model="form.password"/></el-form-item><el-form-item><el-button type="primary" v-on:click="onSubmit('loginForm')">登录</el-button></el-form-item></el-form><el-dialogtitle="温馨提示":visible.sync="dialogVisible"width="30%":before-close="handleClose"><span>请输入账号和密码</span><span slot="footer" class="dialog-footer"><el-button type="primary" @click="dialogVisible = false">确 定</el-button></span></el-dialog></div>

</template><script>export default {name: "Login",data() {return {form: {username: '',password: ''},// 表单验证，需要在 el-form-item 元素中增加 prop 属性rules: {username: [{required: true, message: '账号不可为空', trigger: 'blur'}],password: [{required: true, message: '密码不可为空', trigger: 'blur'}]},// 对话框显示和隐藏dialogVisible: false}},methods: {onSubmit(formName) {// 为表单绑定验证功能this.$refs[formName].validate((valid) => {if (valid) {// 使用 vue-router 路由到指定页面，该方式称之为编程式导航this.$router.push("/main");} else {this.dialogVisible = true;return false;}});}}}

</script><style lang="scss" scoped>.login-box {border: 1px solid #DCDFE6;width: 350px;margin: 180px auto;padding: 35px 35px 15px 35px;border-radius: 5px;-webkit-border-radius: 5px;-moz-border-radius: 5px;box-shadow: 0 0 25px #909399;}.login-title {text-align: center;margin: 0 auto 40px auto;color: #303133;}

</style>

创建一个router目录，编写index.js文件：

import Vue from 'vue'

import Router from 'vue-router'//添加路由

import Main from '../views/Main'

import Login from '../views/Login'Vue.use(Router);export default new Router({routes: [{path: '/main',component: Main},{path: '/login',component: Login}]

})

在main.js中编写：

import Vue from 'vue'

import App from './App'import router from './router'//导入elementUI

import ElementUI from 'element-ui'

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

Vue.use(ElementUI)Vue.config.productionTip = falsenew Vue({el: '#app',router,render: h => h(App) //ElementUI

})

最后App.vue中代码如下：

<template><div id="app"><router-link to="/login">登录页</router-link><router-view></router-view></div>

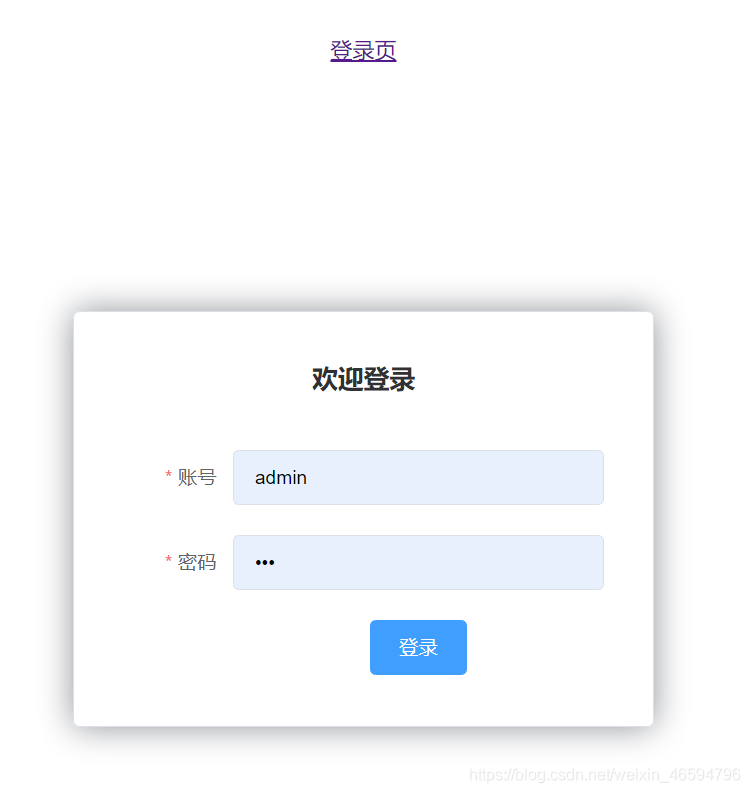
</template><script>export default {name: 'App'

}

</script><style>#app {font-family: 'Avenir', Helvetica, Arial, sans-serif;-webkit-font-smoothing: antialiased;-moz-osx-font-smoothing: grayscale;text-align: center;color: #2c3e50;margin-top: 60px;}

</style>

效果如下：



**十八、路由嵌套**

​ 嵌套路由又称子路由，在实际应用中，通常由多层嵌套的组件组合而成。

在 views/user 目录下创建一个名为 Profile.vue 的视图组件：

<template><h1>个人信息</h1>

</template><script>export default {name: "UserProfile"}

</script><style scoped></style>

在用户列表组件在 views/user 目录下创建一个名为 List.vue 的视图组件：

<template><h1>用户列表</h1>

</template><script>export default {name: "UserList"}

</script><style scoped></style>

修改首页视图，我们修改 Main.vue 视图组件，此处使用了 ElementUI 布局容器组件，代码如下：

<template><div><el-container><el-aside width="200px"><el-menu :default-openeds="['1']"><el-submenu index="1"><template slot="title"><i class="el-icon-caret-right"></i>用户管理</template><el-menu-item-group><el-menu-item index="1-1"><!--插入的地方--><router-link to="/user/profile">个人信息</router-link></el-menu-item><el-menu-item index="1-2"><!--插入的地方--><router-link to="/user/list">用户列表</router-link></el-menu-item></el-menu-item-group></el-submenu><el-submenu index="2"><template slot="title"><i class="el-icon-caret-right"></i>内容管理</template><el-menu-item-group><el-menu-item index="2-1">分类管理</el-menu-item><el-menu-item index="2-2">内容列表</el-menu-item></el-menu-item-group></el-submenu><el-submenu index="3"><template slot="title"><i class="el-icon-caret-right"></i>系统管理</template><el-menu-item-group><el-menu-item index="3-1">用户设置</el-menu-item></el-menu-item-group></el-submenu></el-menu></el-aside><el-container><el-header style="text-align: right; font-size: 12px"><el-dropdown><i class="el-icon-setting" style="margin-right: 15px"></i><el-dropdown-menu slot="dropdown"><el-dropdown-item>个人信息</el-dropdown-item><el-dropdown-item>退出登录</el-dropdown-item></el-dropdown-menu></el-dropdown></el-header><el-main><!--在这里展示视图--><router-view /></el-main></el-container></el-container></div>

</template>

<script>export default {name: "Main"}

</script>

<style scoped lang="scss">.el-header {background-color: #B3C0D1;color: #333;line-height: 60px;}.el-aside {color: #333;}

</style>

配置嵌套路由修改 router 目录下的 index.js 路由配置文件，使用children放入main中写入子模块，代码如下:

import Vue from 'vue'

import Router from 'vue-router'//添加路由

import Main from '../views/Main'

import Login from '../views/Login'import UserList from '../views/user/List'

import UserProfile from '../views/user/Profile'Vue.use(Router);export default new Router({routes: [{path: '/main',component: Main,//嵌套路由children: [{path: '/user/profile',component: UserProfile},{path: '/user/list',component: UserList}]},{path: '/login',component: Login}]

})

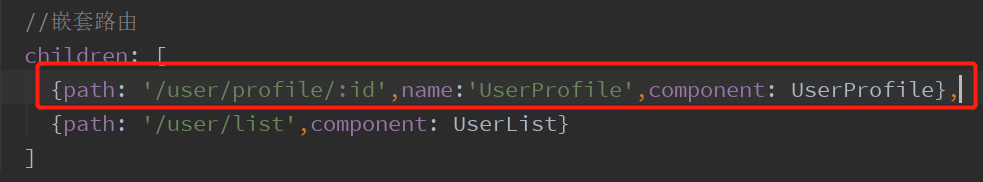
效果展示如下：



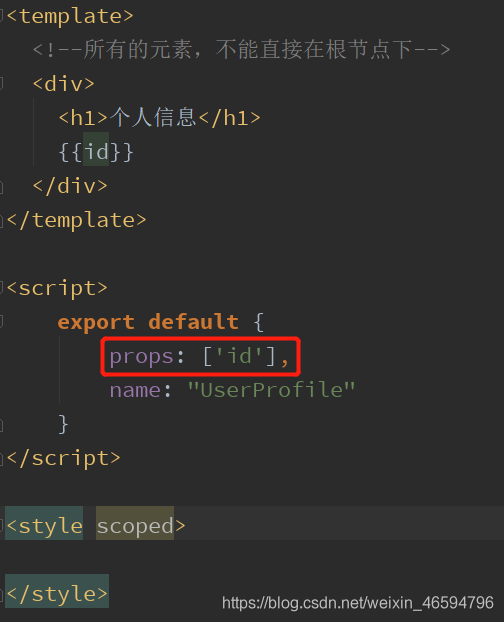
**十九、参数传递**

接下来，还是使用上述代码进行测试！

方式一：取值

1. 修改路由配置, 主要是router下的index.js中的 path 属性中增加了 :id 这样的占位符：
2. 
3. 此时我们在Main.vue中的route-link位置处 to 改为了 :to，是为了将这一属性当成对象使用，注意 router-link 中的 name 属性名称 一定要和 路由中的 name 属性名称 匹配，因为这样 Vue 才能找到对应的路由路径
4. 
5. 在要展示的组件Profile.vue中接收参数 使用 {{$route.params.id}}来接收
6. 

方式二：使用props减少耦合

1. 修改路由配置 , 主要在router下的index.js中的路由属性中增加了 props: true 属性
2. 
3. 传递参数和之前一样 在Main.vue中修改router-link地址
4. 
5. 在Profile.vue接收参数为目标组件增加 props 属性
6. 
7. 效果如下：
8. 

十九、重定向

在router下面index.js的配置：

{path: '/goHome',redirect: '/main'

}

这里定义了两个路径，一个是 /main ，一个是 /goHome，其中 /goHome 重定向到了 /main 路径，由此可以看出重定向不需要定义组件；

使用的话，只需要在Main.vue设置对应路径即可；



**二十、路由模式与处理404**

1.路由模式

路由模式有两种：

* hash：路径带 # 符号，如 http://localhost/#/login
* history：路径不带 # 符号，如 http://localhost/login

修改路由配置，index.js代码如下：



2.处理404

创建于给NotFound.vue的视图组件：

<template><div><h1>404,你的页面走丢了</h1></div>

</template><script>export default {name: "NotFound"}

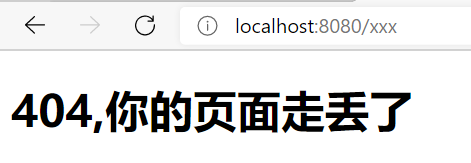
</script><style scoped></style>

修改路由配置，代码如下：

import NotFound from '../views/NotFound'export default new Router({mode: 'history',routes: [{path: '\*',component: NotFound}]

})

当我们输入的地址不正确，会出现以下效果：



**二十一、路由钩子与异步请求**

1.路由钩子函数

**beforeRouteEnter**：在进入路由前执行

**beforeRouteLeave**：在离开路由前执行

在Profile.vue中写:

<template><!--所有的元素，不能直接在根节点下--><div><h1>个人信息</h1>{{id}}</div>

</template><script>export default {props: ['id'],name: "UserProfile",beforeRouteEnter:(to,from,next)=>{console.log("进入路由之前")next();},beforeRouteLeave:(to,from,next)=>{console.log("进入路由之后")next();}}

</script><style scoped></style>

参数说明：

* to：路由将要跳转的路径信息
* from：路径跳转前的路径信息
* next：路由的控制参数
* next() 跳入下一个页面
* next(’/path’) 改变路由的跳转方向，使其跳到另一个路由
* next(false) 返回原来的页面
* next((vm)=>{}) 仅在 beforeRouteEnter 中可用，vm 是组件实例

2.在钩子函数中使用异步通信

1. 安装Axios:

cnpm install axios -s

1. main.js引用 Axios

import axios from 'axios'

import VueAxios from 'vue-axios'

Vue.use(VueAxios, axios)

1. 准备数据 ： 只有我们的 static 目录下的文件是可以被访问到的，所以我们就把静态文件放入该目录下。数据和之前用的json数据一样：data.json
2. 在 beforeRouteEnter 钩子函数中进行异步请求，Profile.vue代码如下：

<template><!--所有的元素，不能直接在根节点下--><div><h1>个人信息</h1>{{id}}</div>

</template><script>export default {props: ['id'],name: "UserProfile",beforeRouteEnter:(to,from,next)=>{console.log("进入路由之前")next(vm => {vm.getData(); //进入路由之前执行getData()});},beforeRouteLeave:(to,from,next)=>{console.log("进入路由之后")next();},methods: {getData: function () {this.axios({method: 'get',url: 'http://localhost:8080/static/mock/data.json'}).then(function (response) {console.log(response)})}}}

</script><style scoped></style>

效果如下：

