**【****Vue】**

# 【主要内容】

1. vue简介
2. M V VM的思想
3. VS Code工具的使用
4. 常用指令
5. 属性，事件，样式绑定
6. 条件渲染、列表循环
7. 过滤器的使用
8. 生命周期
9. 监听器
10. 购物车和医疗处方案例
11. 组件和路由的使用
12. node、npm、vuecli
13. elementui
14. axios网络请求的封装和使用
15. vue-element-admin前端框架的使用

# 【学习目标】

|  |  |
| --- | --- |
| 知识点 | 要求 |
| vue简介 | 熟悉 |
| MVVM的思想 | 熟悉 |
| VS Code工具的使用 | 掌握 |
| 常用指令 | 掌握 |
| 属性，事件，样式绑定 | 掌握 |
| 条件渲染、列表循环 | 掌握 |
| 过滤器的使用 | 熟悉 |
| 生命周期 | 熟悉 |
| 监听器 | 掌握 |
| 购物车和医疗处方页面的案例 | 掌握 |
| 组件和路由的使用 | 掌握 |
| node和npm | 掌握 |
| elementui | 掌握 |
| axios网络请求的封装和使用 | 掌握 |
| vue-element-admin前端框架的使用 | 掌握 |

# 【熟悉】Vue简介

## 简介

它是一个构建用户界面的框架

是一个轻量级的MVVM（Model-View-ViewModel），其实就是所谓的数据的双向绑定

ajax 要面页面上面显示某一个通过ajax请求回来的数据

1. 在页面上面声明一个标签可以存放数据 <div id=”data”></div>
2. 发送请求得到数据
3. 根据id找到页面的div标签var div=$(“#data”) var div=document.getElementById(‘data’)
4. 使用返回的数据拼装html代码
5. 再把拼装好的html放到div里面

数据驱动和组件化的前端开发

通过简单的API就能实现响应式的数据绑定和组合的视图组件

指令：

用来扩展html标签的功能

<https://cn.vuejs.org/>

## 其它

简单、易学、更轻量

指令v-xxx开头

HTML + JSON数据

开发者 尤雨溪 华人

这几个框架都不兼容低版本IE

# 【掌握】MVVM的思想

## MVVM 设计思想

**MVC**

**|--M Model （domain,service,serviceimpl.utils.pojo.mapper）**

**|--V view thymeleaf jsp html**

**|--C controller 接收前端请求**

**Model**：对应数据层的域模型，它主要做域模型的同步。通过 Ajax/fetch 等 API 完成客户端和服务端业务 Model 的同步。在模型层间关系里，它主要用于抽象出 ViewModel 中视图的 Model。个人理解：后端提供API，后端服务架构是 控制器+数据模型 或者 纯控制器。

**View**：View是作为视图模板，用于定义结构、布局。它自己不处理数据，只是将ViewModel中的数据展现出来。此外为了和ViewModel产生关联，那么还需要做的就是数据绑定的声明，指令的声明，事件绑定的声明。ViewModel和View之间是双向绑定，意思就是说ViewModel的变化能够反映到View中，View的变化也能够改变ViewModel的数据值。

**ViewModel**：ViewModel起着连接View和Model的作用，同时用于处理View中的逻辑。在MVC框架中，视图模型通过调用模型中的方法与模型进行交互，然而在MVVM中View和Model并没有直接的关系，在MVVM中，ViewModel从Model获取数据，然后应用到View中。个人理解：Web前端的webserver对View进行双向绑定渲染。

整个MVVM实际上实现了**前后端分离**，通过api来实现前后端交互，前端通过纯js或者双向绑定框架来渲染页面。

**大概如：**

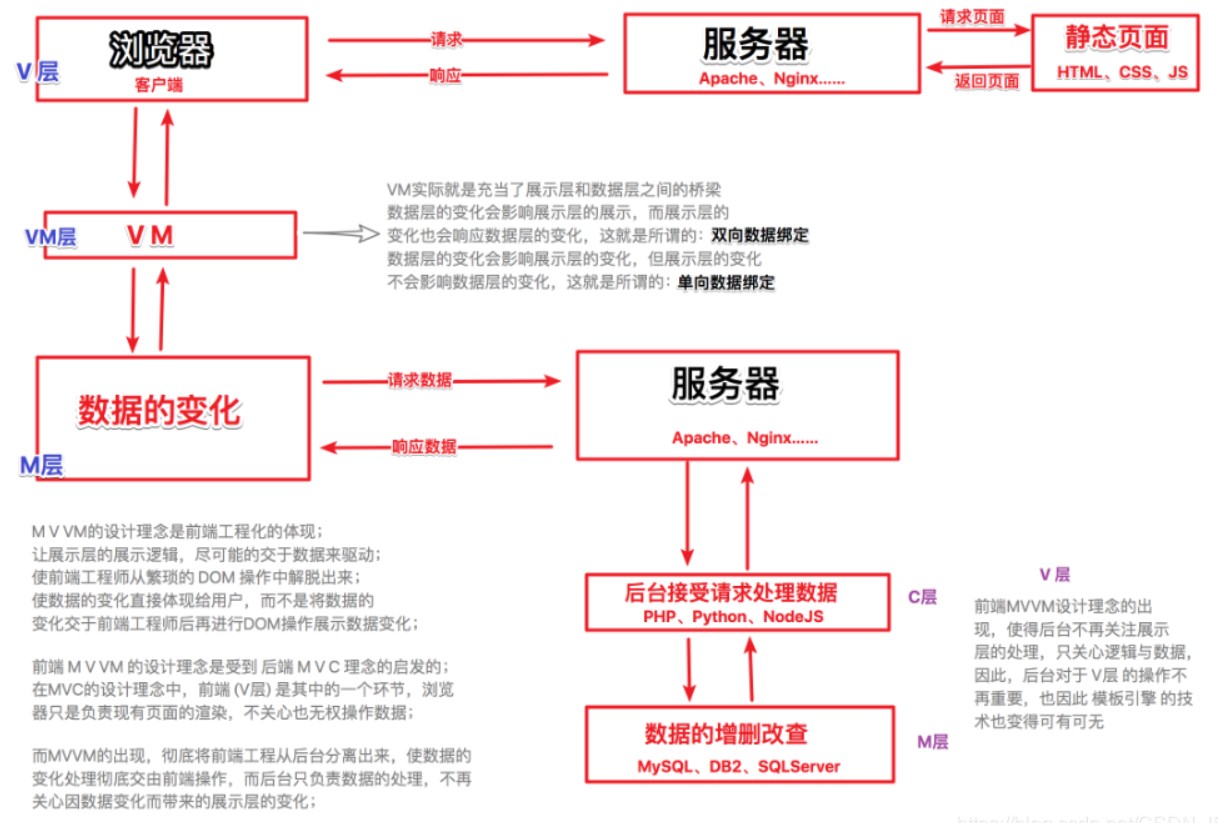
数据库（MySQL、PostgreSQL）<—双向交互—>api（php、java、Python、node）<—双向交互—>ajax/fetch/websocket（node服务、jQ、js）<—双向绑定—>html（标签、css）。

MVVM有利于项目分工和升级，所谓对前后端分离。但也有缺点，就是不利于 。

MVC：服务端来渲染数据，老旧模式。MC属于纯后端，V属于前端，js权重不高，有利于SEO。

万物基于api，一套api可以针对小程序、app、前端，为何不首先使用。需要SEO对部分，单独分离出项目，采用MVC渲染静态页面或者纯html即可。

## MVVM图例解析



# 【掌握】Visual Studio Code的使用

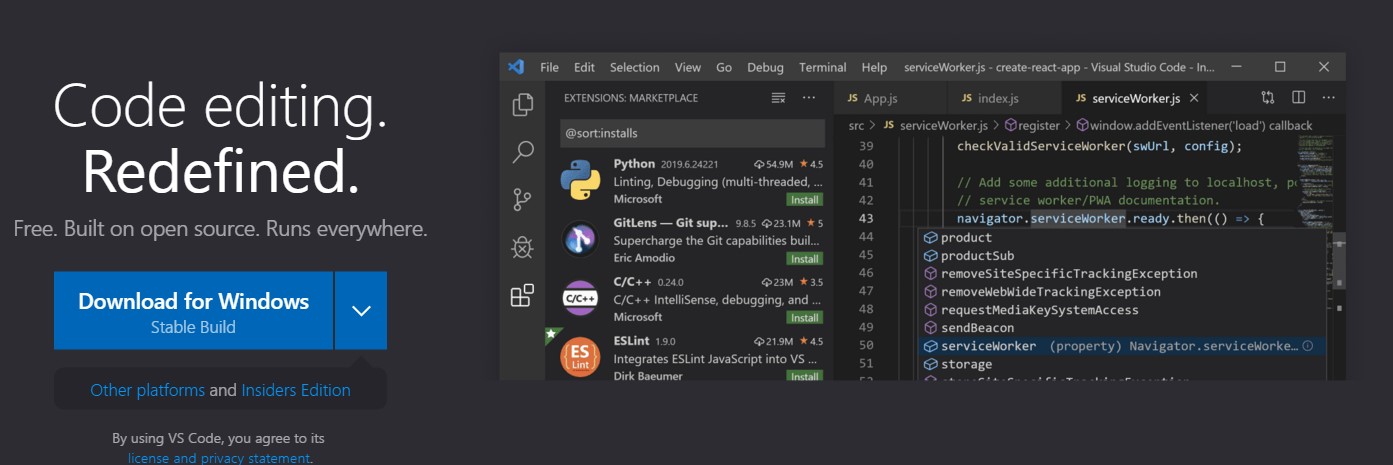
## 为什么使用 VS Code

**HbuilderX**

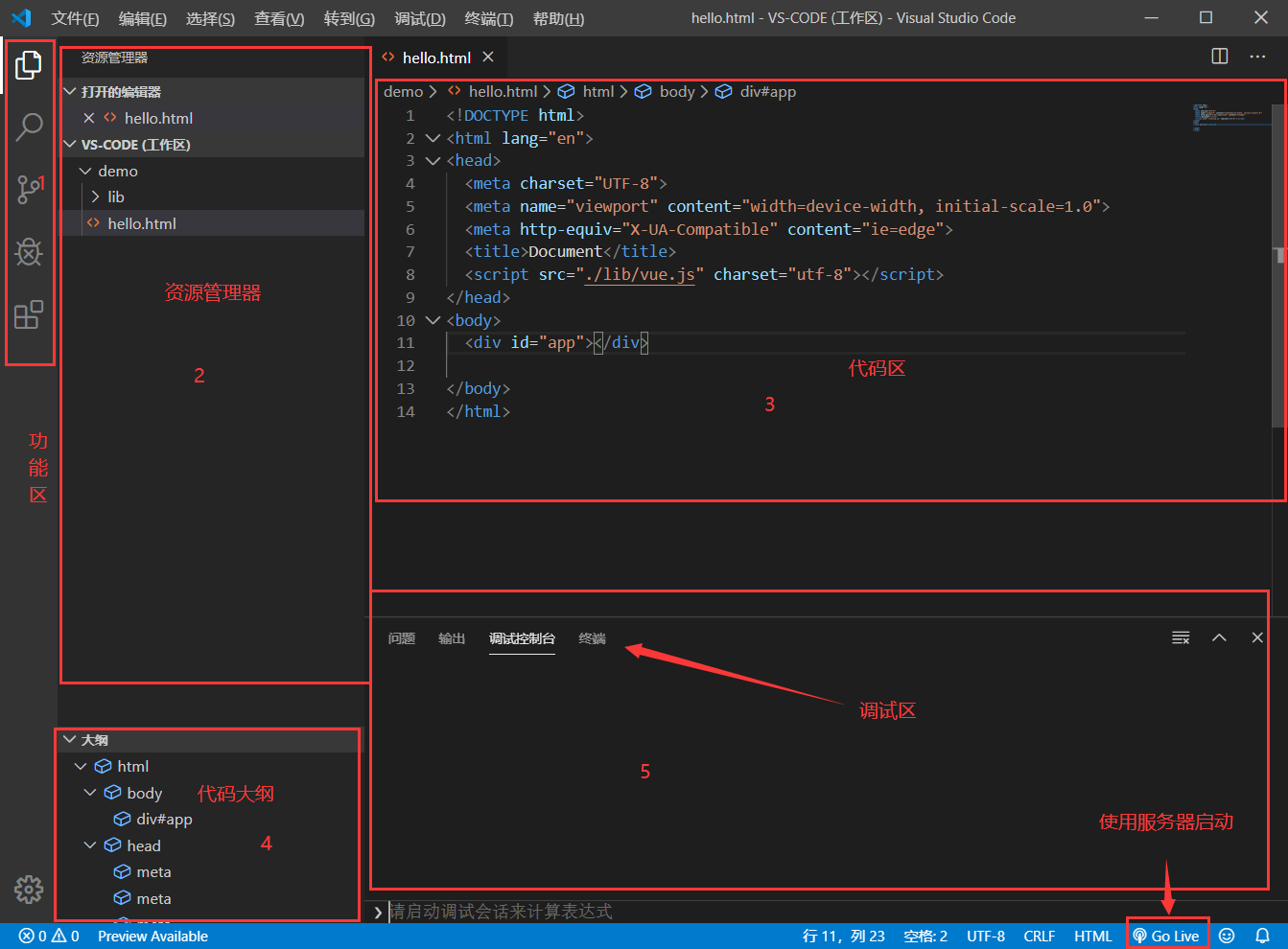
    以JavaScript为代表，过去我们调试JS的时候，需要使用浏览器F12，代码上面需要利用console.log 或 alert 对过程进行输出，习惯了例如VS 编译器的开发人员会比较不习惯。如今可以使用VS Code进行开发，它开源免费，通过安装开发语言相关的插件，可以让VS Code实现相应的语法识别和代码提示，目前拓展商店已经提供了大多数编程语言的插件，可以随便下载，常用有Script有JScript、EScript ,以及包括PHP，Python等其它语言。

## VS Code 安装

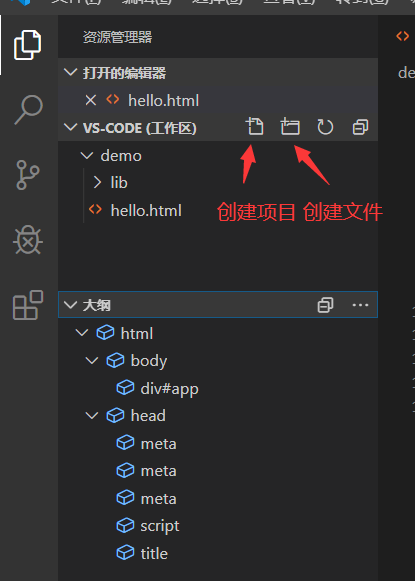
Visual Studio Code （这里简称 VS Code） 下载地址： <https://code.visualstudio.com/>

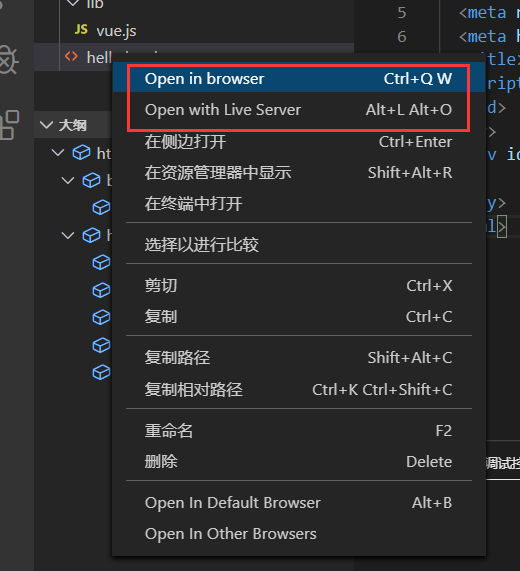


### 面板介绍



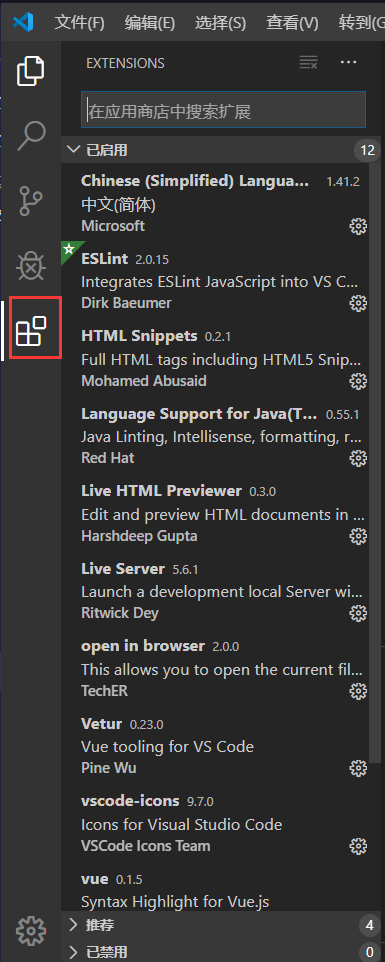
## 项目创建并运行





## 插件安装【直接使用我给的json配置】

### 安装位置



### 常用插件安装

HTML Snippets： 超级实用且初级的 H5代码片段以及提示

HTMLHint： html代码检测

HTML CSS Support : 让 html 标签上写class 智能提示当前项目所支持的样式。新版已经支持scss文件检索，这个也是必备插件之一

Path Autocomplete : 路径智能补全

Path Intellisense ： 路径智能提示

JavaScript Snippet Pack： 针对js的插件，包含了js的常用语法关键字，很实用；

View InBrowser： 从浏览器中查看html文件，使用系统的当前默认浏览器

Class autocomplete for HTML： 编写html代码的朋友们对html代码的一大体现就是重复，如果纯用手敲不仅累还会影响项目进度，这款自动补全插件真的很棒；

beautify : 格式化代码的工具，可以格式化JSON|JS|HTML|CSS|SCSS,比内置格式化好用

Debugger for Chrome： 让 vscode 映射 chrome 的 debug功能，静态页面都可以用 vscode 来打断点调试，真666~

jQuery Code Snippets： jquery 重度患者必须品

vscode-icon： 让 vscode 资源树目录加上图标，必备良品！

VSCode Great Icons： 另一款资源树目录图标

Code Runner : 代码编译运行看结果，支持众多语言

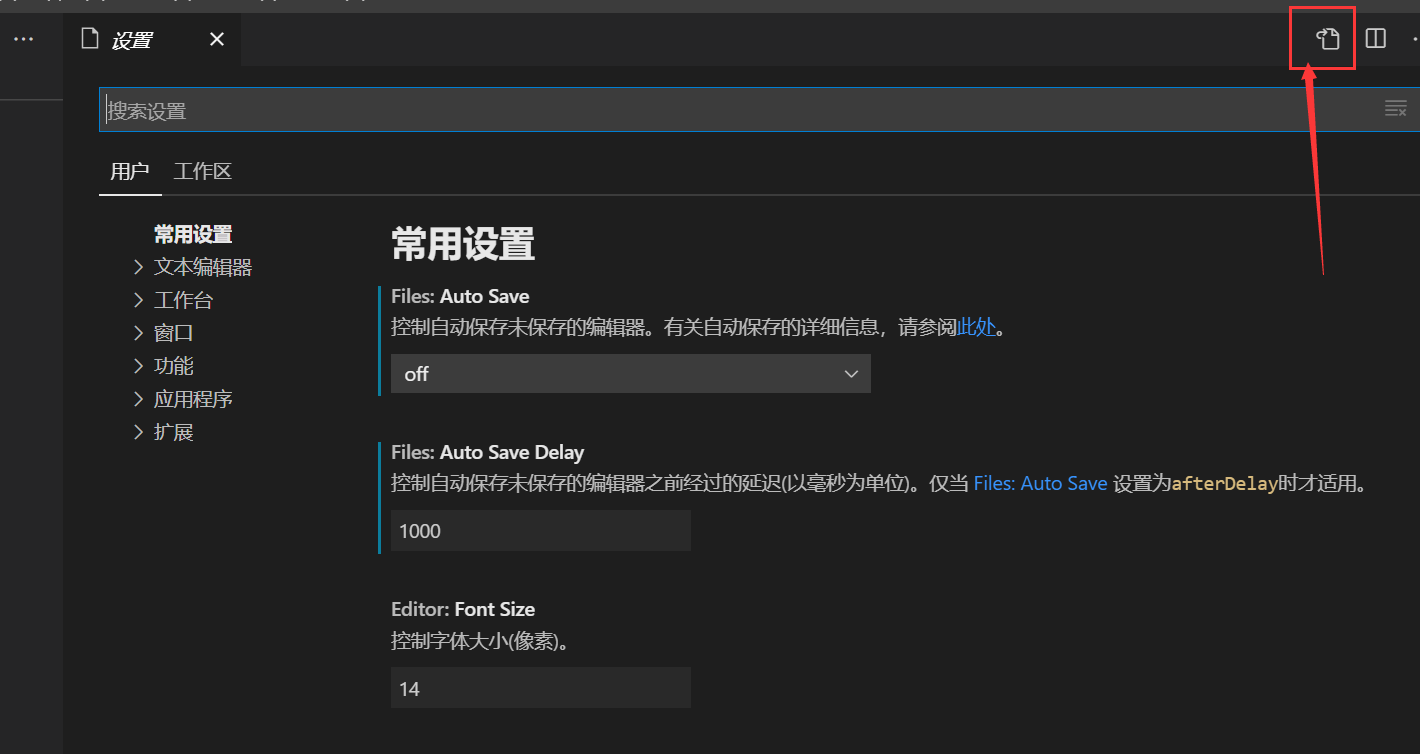
GitLens： 丰富的git日志插件

vetur： vue语法高亮、智能感知、Emmet等

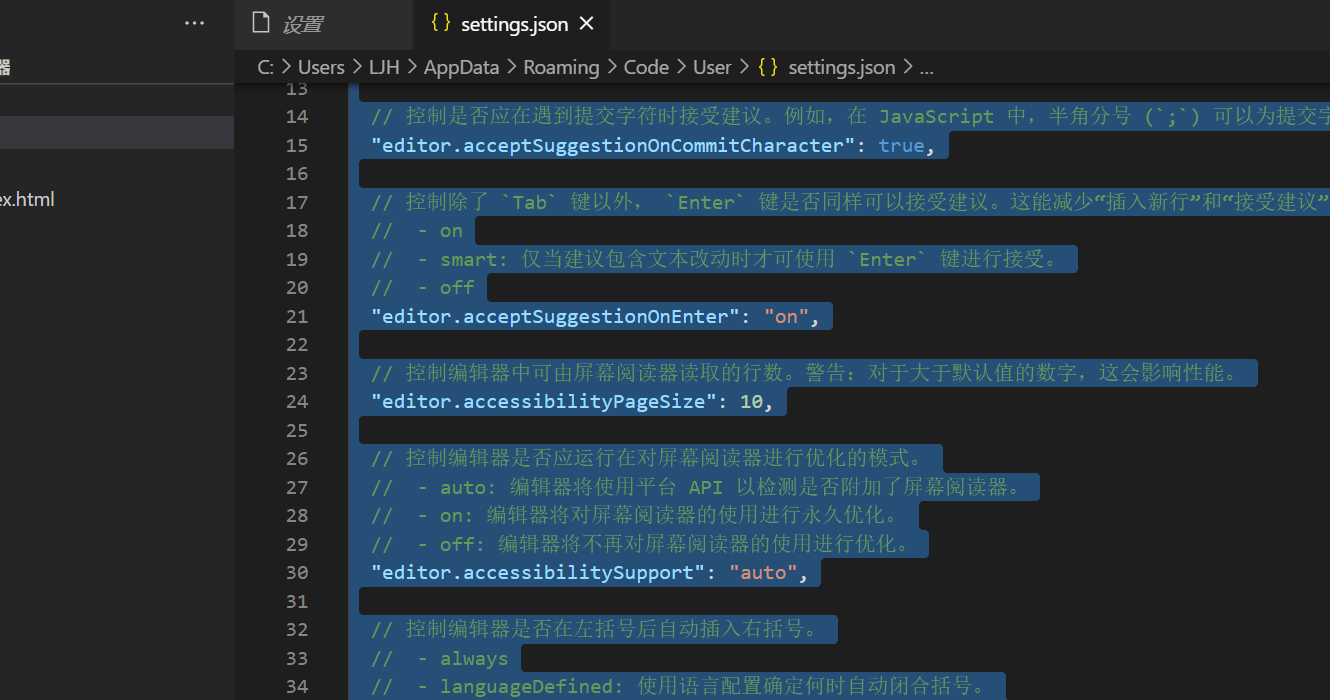
VueHelper： vue代码提示

### 导入雷哥的配置文件【先把插件装好】

**文件-->首选项-=>设置-->**

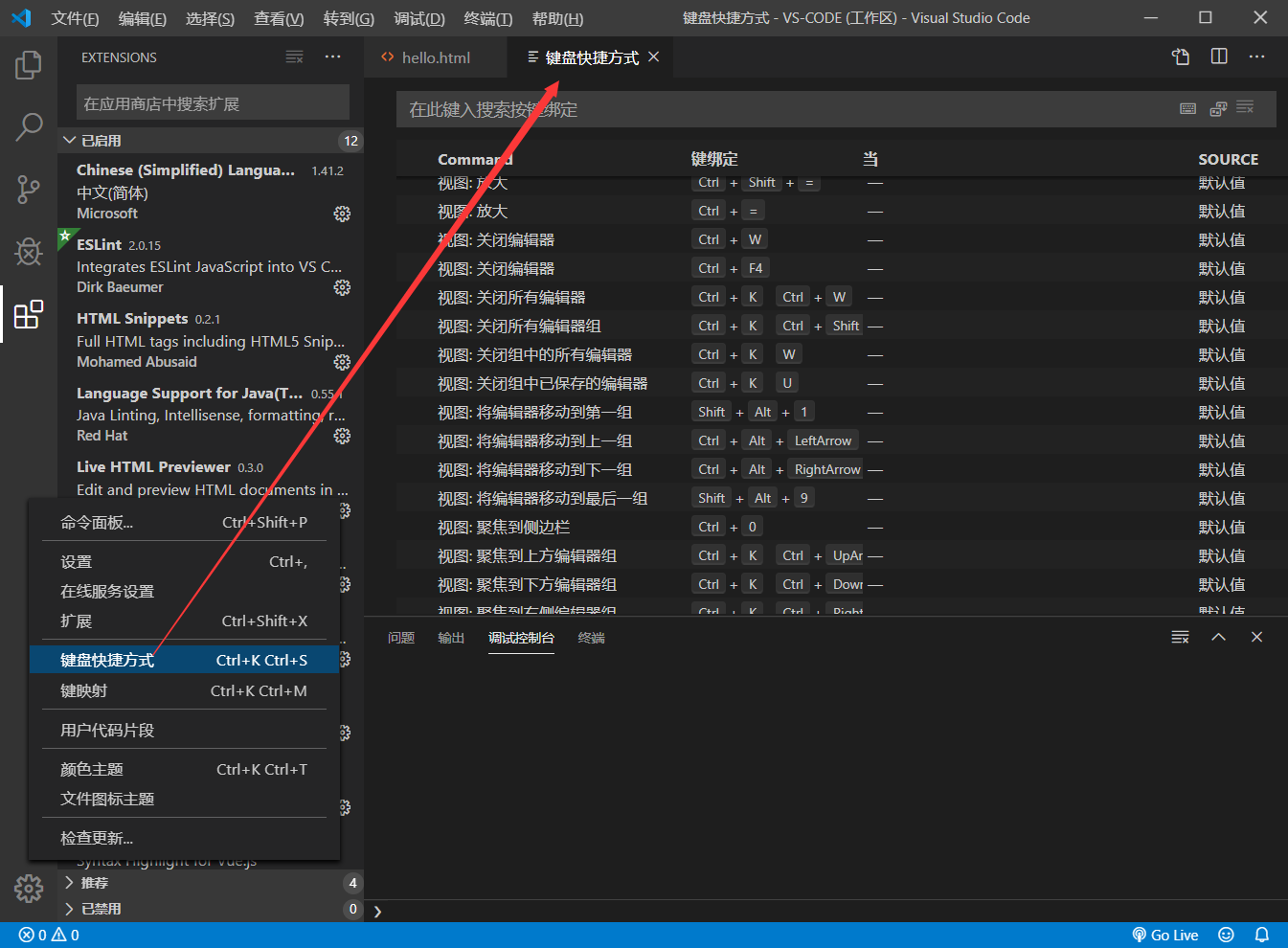


把给的配置文件里面的所有东西覆盖到这个配置文件里面



## 快捷键使用

### **查看快捷键并设置**



### **常用快捷键说明**

**通用**

Ctrl+Shift+P, F1 打开命令面板

Ctrl+P 快速打开文件

Ctrl+Shift+N 打开新窗口/实例

Ctrl+Shift+W 关闭窗口/实例

Alt+Shift+R 打开所在文件夹

**Alt+Shift+F 格式化代码**

**基础编辑**

**Ctrl+X 剪切当前行**

**Ctrl+C 复制当前行**

**Alt+ ↑ / ↓ 向上/向下移动当前行**

**Shift+Alt + ↓ / ↑ 向上/向下复制当前行**

Ctrl+Shift+K 删除当前行 ===使用ctrl+x替换

Ctrl+Enter 在当前行以下插入

Ctrl+Shift+Enter 在当前行以上插入

Ctrl+Shift+\ 跳转到匹配的括号

**Ctrl+/ 添加/删除行注释**

**Shift+Alt+A 添加/删除块注释**

**导航**

Ctrl+T 显示所有符号

Ctrl+G 跳转到行

Ctrl+P 跳转到文件

Ctrl+Shift+O 跳转到符号

Ctrl+Shift+M 显示问题面板

F8 跳转到下一个问题或警告

Shift+F8 跳转到前一个问题或警告

Ctrl+Shift+Tab 显示编辑器文件历史

Alt+ ← / → 向后/向前查看文件

Ctrl+M 开启/关闭 Tab 移动焦点

**搜索和替换**

**Ctrl+F 查找**

**Ctrl+H 替换**

F3 / Shift+F3 查找下一个/前一个

Alt+Enter 选择所有匹配项

Ctrl+D 选择下一个匹配项

Ctrl+K Ctrl+D 跳过当前选择项

Alt+C / R / W 切换大小写敏感/正则表达式/全词

**编辑管理**

Ctrl+F4, Ctrl+W 关闭编辑的文件

Ctrl+K F 关闭文件夹

Ctrl+\ 拆分编辑器窗口

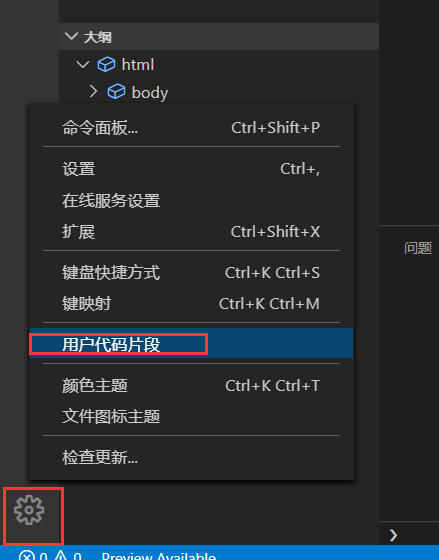
Ctrl+ 1 / 2 / 3 切换到第一、第二或第三个窗口

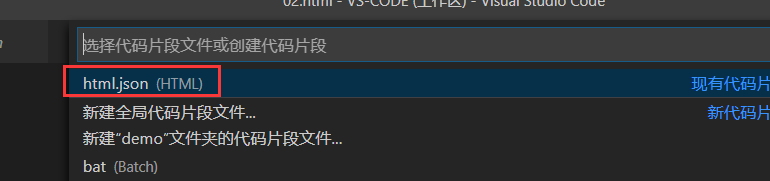
Ctrl+K Ctrl+ ←/→ 切换到上一个/下一个窗口

Ctrl+Shift+PgUp /PgDown 向左/向右移动编辑的文件

Ctrl+K ← / → 向左/向右移动编辑窗口

## html模板使用





// {

//  // Place your snippets for html here. Each snippet is defined under a snippet name and has a prefix, body and

//  // description. The prefix is what is used to trigger the snippet and the body will be expanded and inserted. Possible variables are:

//  // $1, $2 for tab stops, $0 for the final cursor position, and ${1:label}, ${2:another} for placeholders. Placeholders with the

//  // same ids are connected.

//  // Example:

//  // "Print to console": {

//  //  "prefix": "log",

//  //  "body": [

//  //    "console.log('$1');",

//  //    "$2"

//  //  ],

//  //  "description": "Log output to console"

//  // }

// }

{

"h5 template": {

  "prefix": "vh", // 对应的是使用这个模板的快捷键

  "body": [

    "<!DOCTYPE html>",

    "<html lang=\"en\">",

    "<head>",

    "\t<meta charset=\"UTF-8\">",

    "\t<meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">",

    "\t<meta http-equiv=\"X-UA-Compatible\" content=\"ie=edge\">",

    "\t<title>Document</title>",

    "\t<script src=\"./lib/vue-2.4.0.js\"></script>",

    "</head>\n",

    "<body>",

    "\t<div id =\"app\"> </div>\n",

    "\t<script>",

    "\t //创建Vue实例,得到 ViewModel",

    "\t var vm = new Vue({",

    "\t\tel: '#app',",

    "\t\tdata: {},",

    "\t\tmethods: {}",

    "\t });",

    "\t</script>",

    "</body>\n",

    "</html>"

  ],

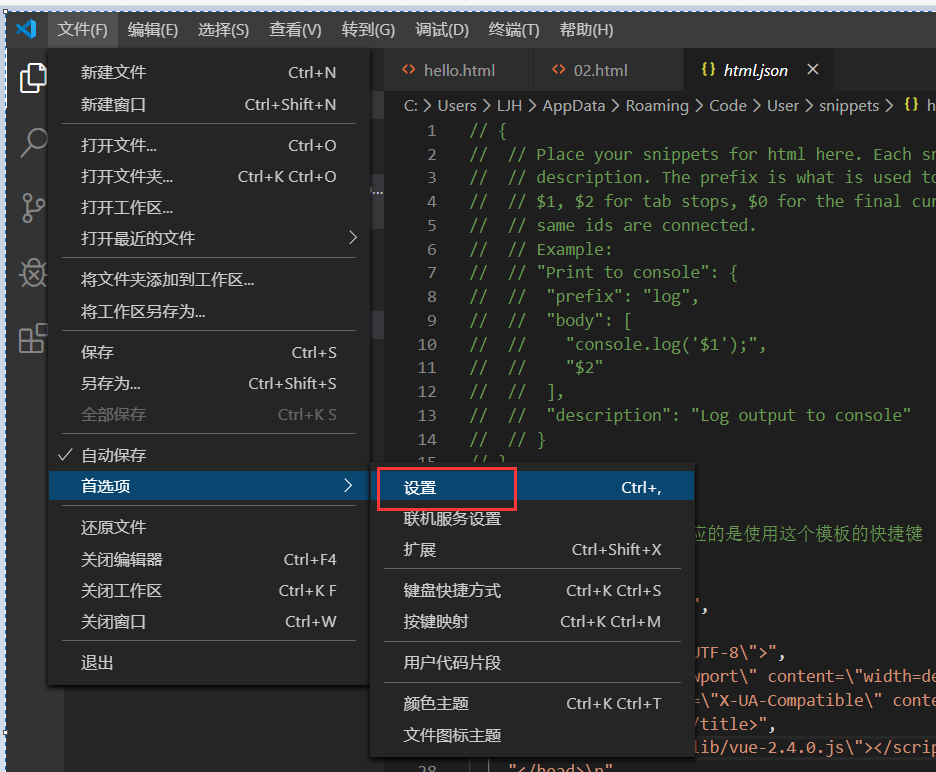
  "description": "HT-H5" // 模板的描述

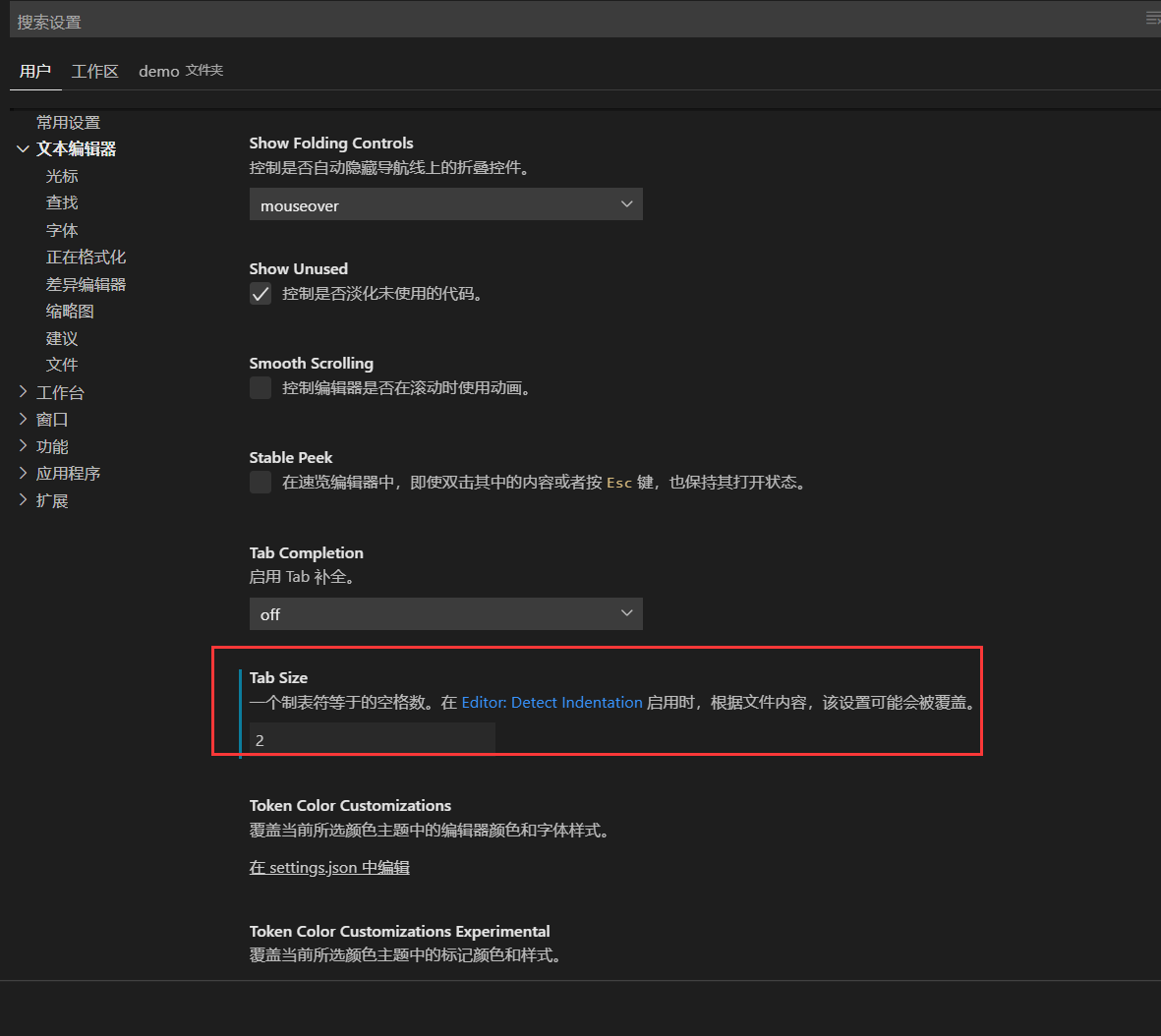
 }

}

## 其它设置

### 默认缩进





### CPU100%

　　有时，vscode会出现CPU利用率100%的情况，两个rg.exe占用了全部的CPU。解决办法如下

　　文件>首选项>设置, 搜索设置 "search.followSymlinks" ：false；

# 【掌握】安装和部署

## 直接下载并用 <script> 标签引入

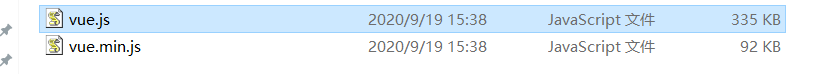
Vue 会被注册为一个全局变量。

在开发环境下不要使用压缩版本，不然你就失去了所有常见错误相关的警告!

开发版本包含完整的警告和调试模式

<https://cn.vuejs.org/v2/guide/installation.html>





## CDN

对于制作原型或学习，你可以这样使用最新版本：

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

对于生产环境，我们推荐链接到一个明确的版本号和构建文件，以避免新版本造成的不可预期的破坏：

<script src="https://cdn.jsdelivr.net/npm/vue@2.6.11"></script>

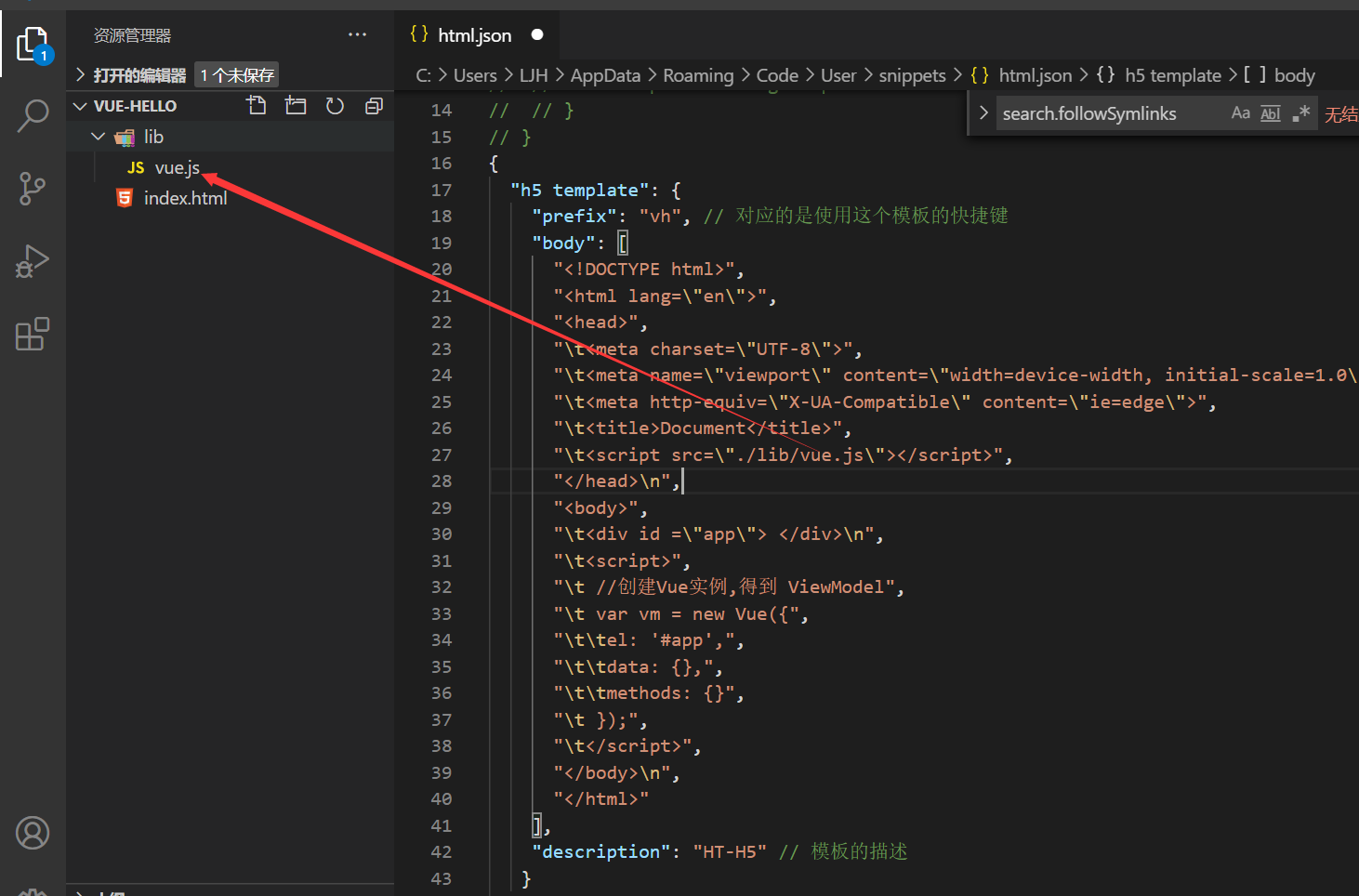
## NPM【后面讲】

在用 Vue 构建大型应用时推荐使用 NPM 安装[1]。NPM 能很好地和诸如 webpack 或 Browserify 模块打包器配合使用。同时 Vue 也提供配套工具来开发单文件组件。

# 最新稳定版

$ npm install vue

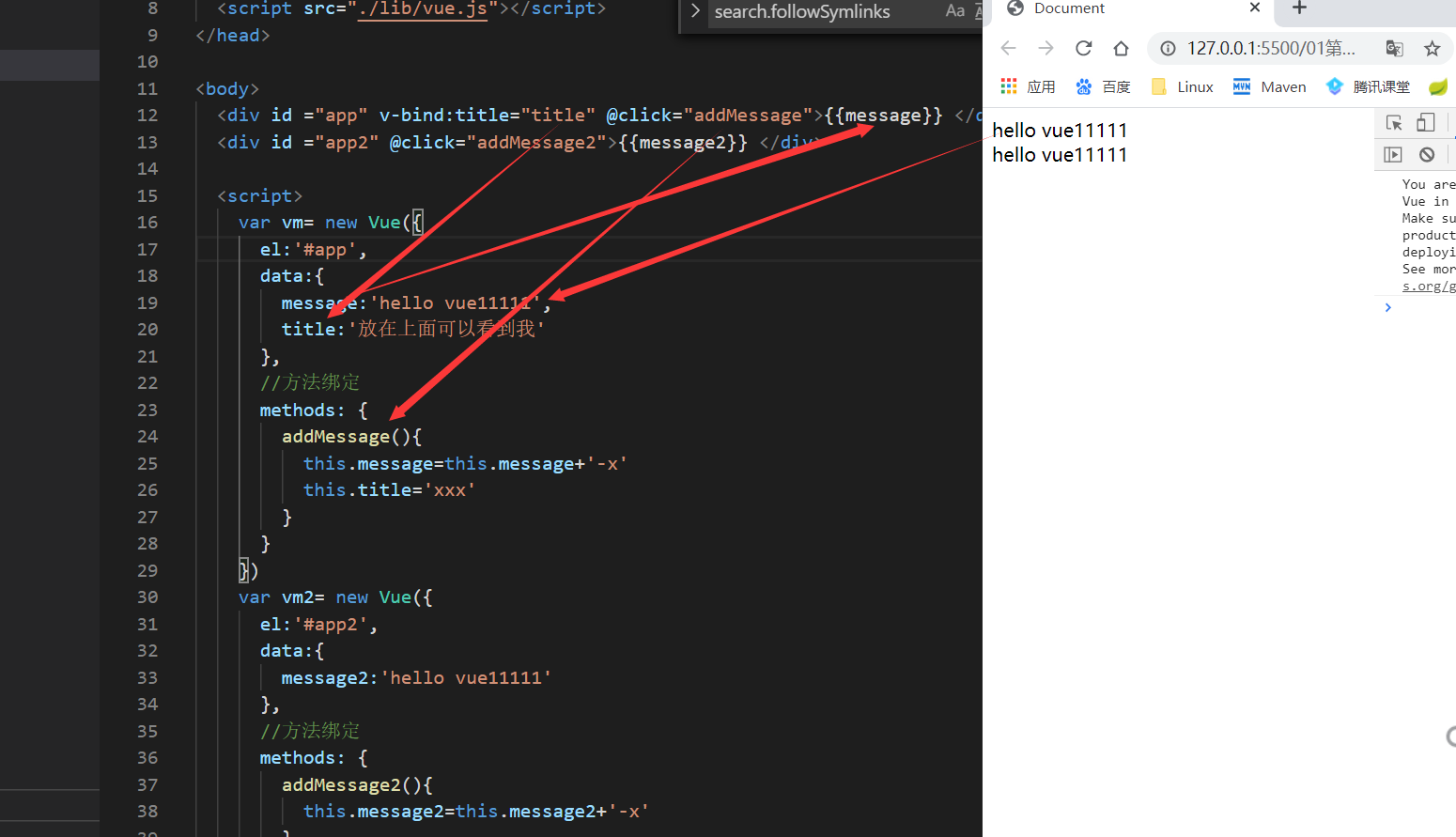
## 入项目并修改模板



|  |
| --- |
| {    "VUE template": {      "prefix": "vh", // 对应的是使用这个模板的快捷键      "body": [        "<!DOCTYPE html>",        "<html lang=\"en\">",        "<head>",        "\t<meta charset=\"UTF-8\">",        "\t<meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">",        "\t<meta http-equiv=\"X-UA-Compatible\" content=\"ie=edge\">",        "\t<title>Document</title>",        "\t<script src=\"./lib/vue.js\"></script>",        "</head>\n",        "<body>",        "\t<div id =\"app\"> </div>\n",        "\t<script>",        "\t //创建Vue实例,得到 ViewModel",        "\t var vm = new Vue({",        "\t\tel: '#app',",        "\t\tdata: {},",        "\t\tmethods: {}",        "\t });",        "\t</script>",        "</body>\n",        "</html>"      ],      "description": "Vue-模板" // 模板的描述     }  } |

## 第一个Vue应用HelloWorld

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>  </head>  <body>    <div id ="app" v-bind:title="title" @click="addMessage">{{message}} </div>    <div id ="app2" @click="addMessage2">{{message2}} </div>    <script>      var vm= new Vue({        el:'#app',        data:{          message:'hello vue11111',          title:'放在上面可以看到我'        },        //方法绑定        methods: {          addMessage(){            this.message=this.message+'-x'            this.title='xxx'          }        }      })      var vm2= new Vue({        el:'#app2',        data:{          message2:'hello vue11111'        },        //方法绑定        methods: {          addMessage2(){            this.message2=this.message2+'-x'          }        }      })    </script>    </body>  </html> |



# 【掌握】v-text,v-html,v-pre,v-once指令

<span id=”aaa”></span>

$(“#aaa”).html(‘<font color=red>RED</font>’)

$(“#aaa”).text(‘<font color=red>RED</font>’)

## 概述

Vue.js 使用了基于 HTML 的模板语法，允许开发者声明式地将 DOM 绑定至底层 Vue 实例的数据。所有 Vue.js 的模板都是合法的 HTML ，所以能被遵循规范的浏览器和 HTML 解析器解析。

在底层的实现上，Vue 将模板编译成虚拟 DOM 渲染函数。结合响应系统，Vue 能够智能地计算出最少需要重新渲染多少组件，并把 DOM 操作次数减到最少。

## v-text

v-text主要用来更新textContent，可以等同于JS的text属性。

 <span v-text="msg"></span>

这两者等价：

 <span>{{msg}}</span>

## v-html

双大括号的方式会将数据解释为纯文本，而非HTML。为了输出真正的HTML，可以用v-html指令。它等同于JS的innerHtml属性。

或者jquery的$().html(aaa)

 <div v-html="rawHtml"></div>

这个div的内容将会替换成属性值rawHtml，直接作为HTML进行渲染。

## v-pre

v-pre主要用来跳过这个元素和它的子元素编译过程。可以用来显示原始的Mustache标签。跳过大量没有指令的节点加快编译。

|  |
| --- |
| <div id="app">  <span v-pre>{{message}}</span> //这条语句不进行编译  <span>{{message}}</span>  </div> |

最终仅显示第二个span的内容

## v-once

v-once关联的实例，只会渲染一次。之后的重新渲染，实例极其所有的子节点将被视为静态内容跳过，这可以用于优化更新性能。

|  |
| --- |
| <span v-once>This will never change:{{msg}}</span> //单个元素  <div v-once>//有子元素      <h1>comment</h1>      <p>{{msg}}</p>  </div> |

上面的例子中，msg即使产生改变，也不会重新渲染。

# 【掌握】属性绑定、事件绑定、双向绑定

涉及指令

1，v-bind  字面意思为绑定。是vue中提供的用户绑定属性的指令。v-bind可简写成:

2，v-on  事件绑定指令 可以简写成@

3，v-model  只能用于表单元素的双向绑定指令

## 属性绑定

在以前的开发中。我们使用title属性去设置鼠标悬浮的显示值，在vue 中可以使用V-bind去动态设置属性，以title为例

|  |
| --- |
| <body>  <div id ="app" >  <!--这里面title可以换成任意属性-->  <span v-bind:title="message">这个将不会改变: {{ message }}</span>11111  </div>  <script>  //创建Vue实例,得到 ViewModel  var vm = new Vue({  el: '#app',  data: {message:'hello vue.js'},  methods: {}  });  </script>  </body> |

## 事件绑定

在以前的开发中。我们使用onclick等属性去设置点击事件，在vue 中可以使用v-on去设置方式，可简写成@

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>  </head>  <body>    <div id ="app">      <img :src="src"  width="200" height="200"  v-bind:title="title" />      <img src="images/bmw.png" width="250" height="250" :title="title" />      <div>{{title}}</div>      <br>      <a href="http://www.baidu.com">去百度</a>      <a :href="baidu">去百度</a>      <br>      <input :value="title" />      <!-- 事件绑定 -->      <img :src="src" v-on:click="toBadu"  width="200" height="200" v-bind:title="title" />      <img :src="src" @click="toBadu"  width="200" height="200" v-bind:title="title" />      <br />      <select @change="chageCity">        <option value="">请选择</option>        <option value="武汉">武汉</option>        <option value="北京">北京</option>        <option value="上海">上海</option>        <option value="广州">广州</option>      </select>      <input type="text" @focus="getFocus" @blur="lostFocus" :value="currentSelectCity">      <div style="width: 300px;height: 300px;background-color: blue;"        @mouseEnter="mouseEnterfun"        @mouseMove="mouseMovefun"        @mouseOut="mouseOutfun"        @mouseDown="mouseDownfun"        @mouseUp="mouseUpfun"      ></div>  <hr>  <input type="text" v-model="msg" />  <div>{{msg}}</div>  <button @click="chageMsg">使用JS改成msg的值</button>    </div>    <script>     //创建Vue实例,得到 ViewModel     var vm = new Vue({      el: '#app',      data: {        title:'BMW',        baidu:"http://www.baidu.com",        src:"images/bmw.png",        currentSelectCity:undefined,        msg:"的是输入框的内容"      },      methods: {        toBadu(){          window.location.href=this.baidu        },        chageCity(x){          console.log(x.srcElement.value);          this.currentSelectCity=x.srcElement.value        },        getFocus(){          console.log("得到焦点")        },        lostFocus(){          console.log("失去焦点")        },        mouseEnterfun(){          console.log("鼠标进入")        },        mouseMovefun(obj){          console.log("鼠标移动x:"+obj.x+" y"+obj.y)        },        mouseOutfun(){          console.log("鼠标出去")        },        mouseDownfun(){          console.log("鼠标按下")        },        mouseUpfun(){          console.log("鼠标弹起")        },        chageMsg(){          this.msg="我是改变之后的值"        }        }     });    </script>  </body>  </html> |

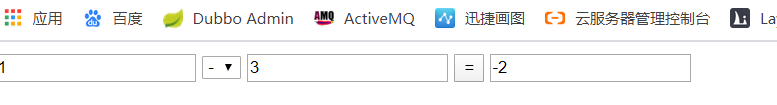
## 双向绑定v-model

v-model  只能用于表单元素的双向绑定指令

v-bind是单向绑定

|  |
| --- |
| <div id ="app" >  <div>{{message}}</div>  <input v-model="message" type="text" />  <button @click="changemsg">changemsg</button>  <div :html="message">{{message}}</div>  </div>    <script>  //创建Vue实例,得到 ViewModel  var vm = new Vue({  el: '#app',  data: {  message:'hello vue.js'  },  methods: {  changemsg(){  this.message="ok"  }  }  });  </script> |

## 双向绑定v-model  小练习



|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>  </head>  <body>    <div id ="app">      <input v-model="number1" />      <select v-model="opt" >        <option value="+">+</option>        <option value="-">-</option>        <option value="\*">\*</option>        <option value="÷">÷</option>      </select>      <input v-model="number2"  />      <button @click="compute">=</button>      <input v-model="cutNumber"/>    </div>    <script>     //创建Vue实例,得到 ViewModel     var vm = new Vue({      el: '#app',      data: {        number1:0,        number2:0,        opt:'+',        cutNumber:0      },      methods: {        compute(){          switch(this.opt){            case '+':              this.cutNumber=parseInt(this.number1)+parseInt(this.number2)              break;            case '-':              this.cutNumber=parseInt(this.number1)-parseInt(this.number2)              break;            case '\*':              this.cutNumber=parseInt(this.number1)\*parseInt(this.number2)              break;            case '÷':              this.cutNumber=parseInt(this.number1)/parseInt(this.number2)              break;          }        }      }     });    </script>  </body>  </html> |

# 【掌握】样式绑定v-bind:class

## 概述

在前端的开发中，我们必须要写的就是css,那么在vue中如何动态的去绑定css呢，请看下面的案例

## 代码

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>    <style>      .red{        color: red;      }      .size{        font-size: 20px;      }      .back{        background-color: blue;        font-size: 20px;      }    </style>  </head>  <body>    <div id ="app">      <!-- 1之前最原始的用法 -->      <div style="background-color: blue;font-size: 18px;color: red;">我是最原始的用法</div>      <!-- 2 可以使用外部样式 -->      <div class="red size back">可以使用外部样式</div>      <!-- 3使用vue的 v-bind去绑定 -->      <div v-bind:class="['red','size']">使用vue的 v-bind去绑定</div>      <!-- 4.使用三元运算符 -->      <div v-bind:class="flag?'red':'size'">使用三元运算符 </div>       <!-- 5.使用对象来代替三元表达式，提高代码可读性 -->      <div :class="{'red':true,'size':true,'back':false}">使用对象来代替三元表达式，提高代码可读性</div>       <!-- 6.通过绑定data中数据来控制style -->      <div :class="classObj">通过绑定data中数据来控制style </div>       <!-- 2和6用的最多。2就是经典的css写法，耦合性也很低。6是基于vue的v-bind的写法，可以动态的改变样式 -->       <button @click="changeFlag">更改flag</button>       <button @click="changeClassObj">更改classObj</button>    </div>    <script>     //创建Vue实例,得到 ViewModel     var vm = new Vue({      el: '#app',      data: {        flag:false,        classObj:{'red':true,'size':true,'back':false}      },      methods: {        changeFlag(){          this.flag=!this.flag        },        changeClassObj(){          this.classObj.red=!this.classObj.red        }      }     });    </script>  </body>  </html> |

# 【掌握】条件渲染v-if v-show

## 相关指令

1，v-if

2，v-else

3，v-else -if

4，v-show

if(){

}else if(){

}else{

}

## v-if

v-if可以实现条件渲染，Vue会根据表达式的值的真假条件来渲染元素。

<div id="app">

<a v-if="ok">yes</a>

</div>

<script>

var vue = new Vue({

el: '#app',

data: {

ok:false,

}

})

</script>

如果属性值ok为true，则显示。否则，不会渲染这个元素。

## v-else

v-else是搭配v-if使用的，它必须紧跟在v-if或者v-else-if后面，否则不起作用。

<a v-if="ok">yes</a>

<a v-else>No</a>

## v-else-if

v-else-if充当v-if的else-if块，可以链式的使用多次。可以更加方便的实现switch语句。

<div id="app">

<div v-if="type==='A'">

A

</div>

<div v-else-if="type==='B'">

B

</div>

<div v-else-if="type==='C'">

C

</div>

<div v-else>

Not A,B,C

</div>

</div>

<script>

var vue = new Vue({

el: '#app',

data: {

ok:false,

type:'A'

}

})

</script>

## v-show

 <h1 v-show="ok">hello world</h1>

也是用于根据条件展示元素。和v-if不同的是，如果v-if的值是false，则这个元素被销毁，不在dom中。但是v-show的元素会始终被渲染并保存在dom中，它只是简单的切换css的dispaly属性。

**注意：v-if有更高的切换开销**

**v-show有更高的初始渲染开销。**

**因此，如果要非常频繁的切换，则使用v-show较好；如果在运行时条件不太可能改变，则v-if较好**

## 案例

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>  </head>  <body>    <div id ="app">      <!-- v-if  v-else 两个标签元素必须是相邻兄弟级别   -->      <div v-if="flag">我是 if true</div>      <!-- <div>sdafasdfasddf</div> -->      <div v-else>我是else</div>        <hr>      <div v-if="score>=90">你很优秀</div>      <div v-else-if="score>=80">良好</div>      <div v-else-if="score>=70">正常</div>      <div v-else-if="score>=60">及格</div>      <div v-else>兄弟努力吧</div>      <input v-model:value="score" />      <!-- v-if  和show是有区别的        1，v-if 如果为false 则，当前这个div根本不会创建  如果为true时会重新创建对象载入        2，v-show 不管为true还是False 先把对象创建出来，再根据flag显示或隐藏      -->      <div v-if="flag">我是 if true</div>      <div v-show="flag">显示吗，显示吧</div>      </div>    <script>     //创建Vue实例,得到 ViewModel     var vm = new Vue({      el: '#app',      data: {        flag:true,        score:100      },      methods: {}     });    </script>  </body>  </html> |

# 【掌握】列表渲染v-for

## v-for

用v-for指令根据遍历数组来进行渲染

有下面两种遍历形式

<div v-for="(item,index) in items"></div> //使用in，index是一个可选参数，表示当前项的索引

<div v-for="item in items"></div> //使用in

下面是一个例子，并且在v-for中，拥有对父作用域属性的完全访问权限。

## v-for案例

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>  </head>  <body>    <div id ="app">      <div v-for="(item,index) in arr1" v-text="item+'-'+index"> {{item}}-{{index}} </div>      <hr>      <div v-for="(item,index) in arr2">        {{index}}-{{item.id}}-{{item.name}}-{{item.address}}      </div>      编号<input type="text"  v-model="id" /><br />      姓名<input type="text"  v-model="name" /><br />      地址<input type="text"  v-model="address" /><br />      <button @click="add">添加</button>    </div>    <script>     //创建Vue实例,得到 ViewModel     var vm = new Vue({      el: '#app',      data: {        arr1:[1,3,4,5,6,7,8],        arr2:[{id:1,name:'小明1',address:'武汉1'},        {id:2,name:'小明2',address:'武汉2'},        {id:3,name:'小明3',address:'武汉3'},        {id:4,name:'小明4',address:'武汉4'},        {id:5,name:'小明5',address:'武汉5'}],        id:undefined,        name:undefined,        address:undefined      },      methods: {        add(){          this.arr2.push({id:this.id,name:this.name,address:this.address})        }      }     });    </script>  </body>  </html> |

## for-layuiCRUD案例



|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>    <link rel="stylesheet" href="./css/layui.css">  </head>  <body>    <div id="app">      <div class="layui-inline">        <div class="layui-input-inline" style="width: 100px;">          <input type="text" placeholder="ID" class="layui-input" v-model="id">        </div>        <div class="layui-input-inline" style="width: 100px;">          <input type="text" placeholder="姓名" class="layui-input" v-model="name">        </div>        <div class="layui-input-inline" style="width: 100px;">          <input type="text" placeholder="性别" class="layui-input" v-model="sex">        </div>        <div class="layui-input-inline" style="width: 100px;">          <input type="text" placeholder="备注" class="layui-input" v-model="comment">        </div>        <div class="layui-input-inline" style="width: 100px;">          <button type="button" class="layui-btn" @click="add">添加</button>        </div>        <div class="layui-input-inline" style="width: 100px;">          <input type="text" placeholder="搜索姓名" class="layui-input" v-model="searchName">        </div>        <div class="layui-input-inline" style="width: 100px;">          <button type="button" class="layui-btn" @click="search">搜索</button>        </div>      </div>      <table class="layui-table">        <colgroup>          <col width="150">          <col width="200">          <col>        </colgroup>        <thead>          <tr>            <th>ID</th>            <th>姓名</th>            <th>性别</th>            <th>备注</th>            <th>操作</th>          </tr>        </thead>        <tbody>          <tr v-for="(item, index) in userList" :key="index">            <td>{{item.id}}</td>            <td>{{item.name}}</td>            <td>{{item.sex}}</td>            <td>{{item.comment}}</td>            <td>              <button class="layui-btn layui-btn-danger" @click="deleteById(item.id)">删除</button>              <button class="layui-btn layui-btn-primary" @click="updateById(item.id)">更新</button>            </td>          </tr>        </tbody>      </table>      <h1>修改用户</h1>      <div class="layui-inline">        <div class="layui-input-inline" style="width: 100px;">          <input type="text" placeholder="ID" class="layui-input" v-model="updateObj.id">        </div>        <div class="layui-input-inline" style="width: 100px;">          <input type="text" placeholder="姓名" class="layui-input" v-model="updateObj.name">        </div>        <div class="layui-input-inline" style="width: 100px;">          <input type="text" placeholder="性别" class="layui-input" v-model="updateObj.sex">        </div>        <div class="layui-input-inline" style="width: 100px;">          <input type="text" placeholder="备注" class="layui-input" v-model="updateObj.comment">        </div>        <div class="layui-input-inline" style="width: 100px;">          <button type="button" class="layui-btn" @click="doUpdate">确定修改</button>        </div>      </div>    </div>    <script>      //创建Vue实例,得到 ViewModel      var vm = new Vue({        el: '#app',        data: {          id: undefined,          name: undefined,          sex: undefined,          comment: undefined,          searchName: undefined,          userList: [            { id: 1, name: '雷哥', sex: '男', comment: '地表最帅男人' },            { id: 2, name: '刘备', sex: '男', comment: '刘知兵' },            { id: 3, name: '关羽', sex: '男', comment: '关过江' },            { id: 4, name: '张飞', sex: '男', comment: '张爱兵' }          ],          updateObj: {            id: undefined,            name: undefined,            comment: undefined,            sex: undefined          }        },        methods: {          add() {            let obj = {              id: this.id,              name: this.name,              sex: this.sex,              comment: this.comment            }            this.userList.push(obj)          },          search() {            let newUserList = new Array();            let itemArr = this.userList.filter(item => {              if (item.name.includes(this.searchName)) {                newUserList.push(item)                return item              }            })            this.userList = newUserList;            console.log(itemArr)          },          deleteById(id) {            // es6语法新增了强大的数组操作函数            let index = this.userList.findIndex(item => {              if (item.id === id) {                return true              }            })            //js根据索引删除数组里面数据的方法 1删除            this.userList.splice(index, 1)          },          updateById(id) {            let itemArr = this.userList.filter(item => {              if (item.id === id) {                return item              }            })            let item = itemArr[0];            this.updateObj = {              id: item.id,              name: item.name,              comment: item.comment,              sex: item.sex            }          },          doUpdate() {            console.log(this.updateObj.id)            let itemArr = this.userList.filter(item => {              if (item.id === this.updateObj.id) {                item.id=this.updateObj.id                item.name=this.updateObj.name                item.sex=this.updateObj.sex                item.comment=this.updateObj.comment              }            })         }        }      });    </script>  </body>  </html> |

# 【掌握】过滤器的使用

## 概述

Vue.js 允许你自定义过滤器，可被用于一些常见的文本格式化。过滤器可以用在两个地方：双花括号插值和 v-bind 表达式 (后者从 2.1.0+ 开始支持)。过滤器应该被添加在 JavaScript 表达式的尾部，由“管道”符号指示：

过滤器分为两种

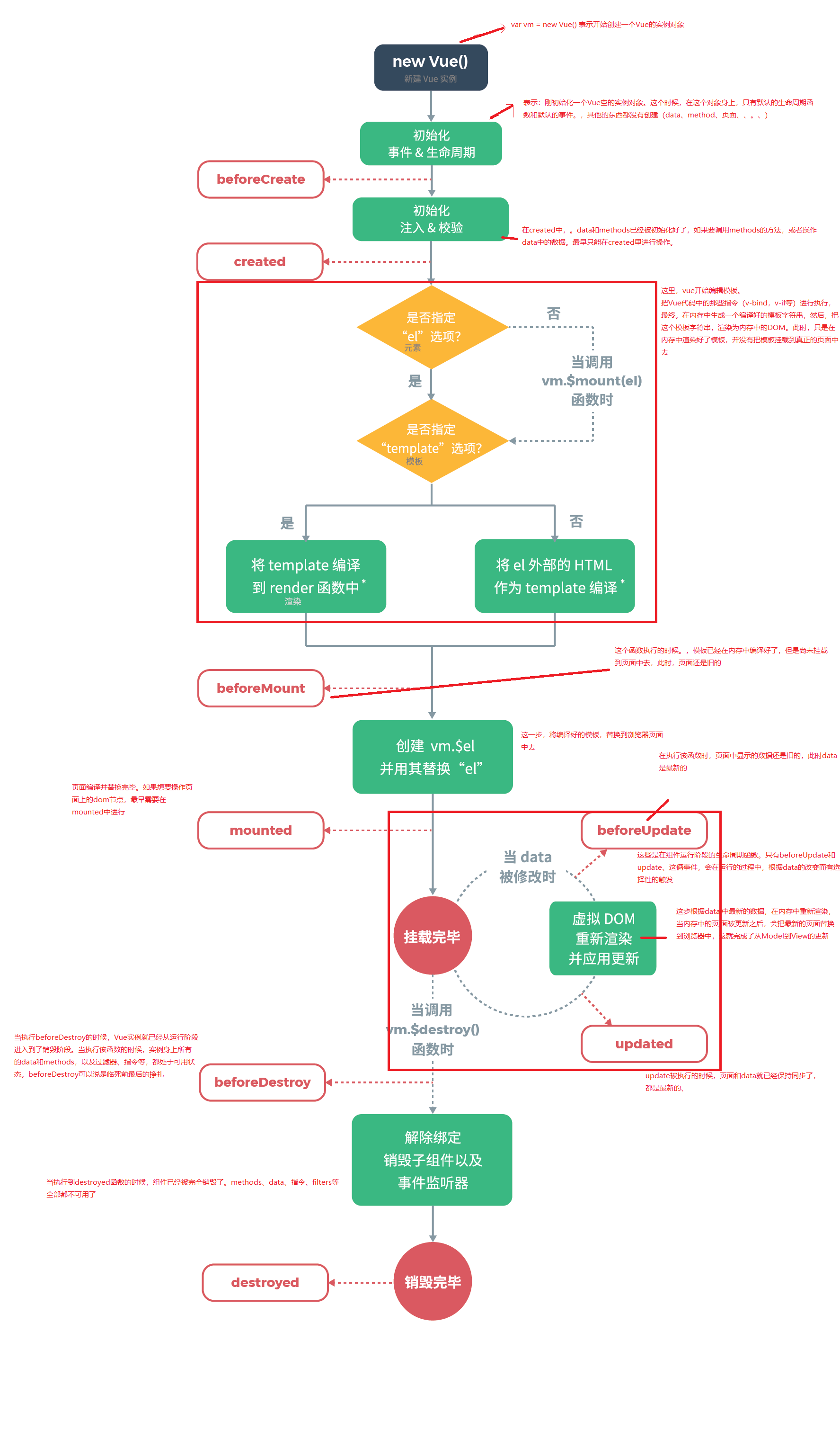
* 全局过滤器 所有Vue实例里面都可以使用
* 局部过滤器 当前的Vue实例里面可以使用

## 代码

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>  </head>  <body>    <div id ="app">      <div v-for="(item,index) in userList">        {{item.id}}        --        {{item.name|handlerName}}        --        {{item.sex|handlerSex}}        --        {{item.salary|handlerSalary}}        --        {{item.birth|handlerBirth}}      </div>    </div>    <hr>    <script>      //定义全局过滤器      // Vue.filter("handlerSex",function(value){      //     return value==1?'男':'女'      // });      //工资保留两位小数的全局过滤器      Vue.filter("handlerSalary",function(value){        return value.toFixed(2)      })      //日期过滤器      // Vue.filter("handlerBirth",function(value){      //     return value.getFullYear()+"-"+(value.getMonth()+1)+"-"+value.getDate()      // });      //特殊字符过滤      // Vue.filter("handlerName",function(value){      //   return value.replace(/4/g,"\*")      // })     //创建Vue实例,得到 ViewModel     var vm1 = new Vue({      el: '#app',      filters :{        // handlerSex:function(value){        //   return value==1?'男':'女'        // },        handlerSex(value){          return value==1?'男':'女'        },        handlerSalary(value){          return value.toFixed(3)        },        handlerBirth(value){          return value.getFullYear()+"-"+(value.getMonth()+1)+"-"+value.getDate()        },        handlerName(value){          return value.replace(/4/g,"\*")        }      },      data: {        userList:[        {id:1,name:'小明111',sex: 1,salary:9999.888,birth:new Date()},        {id:2,name:'小明222',sex: 0,salary:8999.777,birth:new Date()},        {id:3,name:'小明333',sex: 1,salary:7999.666,birth:new Date()},        {id:4,name:'小明444',sex: 0,salary:6999.555,birth:new Date()},        {id:5,name:'小明555',sex: 1,salary:5999.444,birth:new Date()},      ]      },      methods: {        }     });    </script>  </body>  </html> |

# 【掌握】生命周期【重点内容】

## 生命周期图【重点注意create方法】



## 代码

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>  </head>  <body>    <div id ="app">      <span>{{id}}</span>      <br />      <button @click="changeId">修改id</button>    </div>    <script>     //创建Vue实例,得到 ViewModel     var vm = new Vue({      el: '#app',      data: {        id:1      },      beforeCreate() {        console.log("beforeCreate:id:"+this.id)      },      //这个是重点      created() {        console.log("created:id:"+this.id)      },      beforeMount() {        console.log("beforeMount:id:"+this.id)      },      mounted() {        console.log("mounted:id:"+this.id)      },      beforeUpdate() {        console.log("beforeUpdate:id:"+this.id)      },      updated() {        console.log("updated:id:"+this.id)      },      beforeDestroy() {        console.log("beforeDestroy:id:"+this.id)      },      destroyed() {        console.log("destroyed:id:"+this.id)      },      methods: {        changeId(){          this.id=1000        }      }     });    </script>  </body>  </html> |

# 【掌握】监听器

## 作用

监听data里面属性值的变化

会返回原来的值和新的值

## 案例

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>  </head>  <body>    <div id ="app">      <button @click="changeUserName">修改姓名</button>      <button @click="changeSalary">修改工资</button>      <button @click="changeObj">修改对象</button>      <div>{{username}}</div>      <div>        {{userObj}}      </div>    </div>    <script>     //创建Vue实例,得到 ViewModel     var vm = new Vue({      el: '#app',      data: {        username:'小明',        userObj:{          id:1,          name:'小明',          salary:99999.99        }      },      methods: {        changeUserName(){          this.username="习大大"        },        changeSalary(){          this.userObj.salary=999999999        },        changeObj(){          this.userObj={id:2,name:'彭妈妈',salary:12312312}        }      },      watch: {        // userObj(oldObj,newObj){        //   console.log(oldObj)        //   console.log(newObj)        // },        username(oldName,newName){          console.log(oldName+"  "+newName)        },        // 'userObj.salary'(oldName,newName){        //   console.log(oldName+"  "+newName)        // }        'userObj.salary':function(oldName,newName){          console.log(oldName+"  "+newName)        }      },     });    </script>  </body>  </html> |

# 【掌握】购物车

## 效果



## 代码

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <link rel="stylesheet" href="./css/layui.css"  media="all">    <script src="./lib/vue.js"></script>  </head>  <body>    <div id ="app">      <table width="100%" class="layui-table">        <thead>          <tr>            <th>              <input type="checkbox" :checked="isCheckedAll" @click="checkAll" />            </th>            <th>编号</th>            <th>商品名称</th>            <th>单价</th>            <th>数量</th>            <th>小计</th>            <th>备注</th>            <th>操作</th>          </tr>        </thead>        <tbody>          <tr v-for="(item,index) in goodsList">            <td ><input type="checkbox" :checked="item.isChecked" @click="changeChecked(item)"  /></td>            <td >{{item.id}}</td>            <td >{{item.goodsName}}</td>            <td >{{item.price|handlerPrice}}</td>            <td >              <button @click="reduce(item)" >-</button>              {{item.num}}              <button @click="add(item)">-</button>            </td>            <td >{{item.amount|handlerPrice}}</td>            <td >{{item.remark}}</td>            <td >              <button class="layui-btn layui-btn-danger layui-btn-sm" @click="deleteItem(item)">删除</button>            </td>          </tr>          <tr>            <td colspan="1111" align="right">                  应付￥:<span>{{totalAmount|handlerPrice}}</span>                  <button class="layui-btn layui-btn-danger">结算</button>            </td>          </tr>        </tbody>      </table>      <div>{{goodsList}}</div>    </div>    <script>     //创建Vue实例,得到 ViewModel     var vm = new Vue({      el: '#app',      filters:{          handlerPrice(value){            return value.toFixed(2)          }      },      data: {        totalAmount:0.00,        isCheckedAll:false,//是否全选        goodsList:[          {isChecked:false,id:1,goodsName:'小霸王1',price:200,num:1,amount:200,remark:'牛逼'},          {isChecked:false,id:2,goodsName:'小霸王2',price:300,num:1,amount:300,remark:'牛逼'},          {isChecked:false,id:3,goodsName:'小霸王3',price:200,num:1,amount:200,remark:'牛逼'},          {isChecked:false,id:4,goodsName:'小霸王4',price:100,num:1,amount:100,remark:'牛逼'},          {isChecked:false,id:5,goodsName:'小霸王5',price:50,num:1,amount:50,remark:'牛逼'},          {isChecked:false,id:6,goodsName:'小霸王6',price:10,num:1,amount:10,remark:'牛逼'},        ]      },      methods: {        //点击行的checkbox        changeChecked(item){          item.isChecked=!item.isChecked          let allChecked=true          this.goodsList.filter(item=>{             if(!item.isChecked){              allChecked=false             }          })          this.isCheckedAll=allChecked          this.computedTotalAmount()        },        //全选        checkAll(){          this.isCheckedAll=!this.isCheckedAll          this.goodsList.filter(item=>{              item.isChecked= this.isCheckedAll            })            this.computedTotalAmount()        },        //行内减少        reduce(item){          if(item.num>1){            item.num--            item.amount=(item.num\*item.price)            this.computedTotalAmount()          }        },        //行业添加        add(item){          item.num++          item.amount=(item.num\*item.price)          this.computedTotalAmount()        },        //计算结算总额        computedTotalAmount(){          this.totalAmount=0.00;          this.goodsList.filter(item=>{            if(item.isChecked){              this.totalAmount+=item.amount            }          })        },        //删除        deleteItem(obj){          var delIndex=-1;          this.goodsList.filter((item,index)=>{            if(item.id==obj.id){              delIndex=index;            }          })          if(delIndex!=-1){            this.goodsList.splice(delIndex,1);            this.computedTotalAmount()          }        }      }     });    </script>  </body>  </html> |

# 【掌握】组件使用

## 什么是组件

组件的出现，是为了拆分Vue的代码块，不用同的组件划分不同的功能模块，以后我们需要某个功能时，就直接调用对应的组件即可

模块化：模块化是从代码的角度去分析的，方便代码分层开发，保证每个模块职责单一

组件：组件化是从界面的角度去划分的，如：分页组件、轮播、颜色选择、文件上传等

**变量名说明：**

按java的开发思想，变量名往往是**驼峰规则**

在vue中定义组件可以使用驼峰规则，但是使用组件的时候如果存在驼峰，应该全部改成小写，并把每个单词用-连接 有点像springboot的yml配置文件的语法

组件模板的说明

组件的template里面必须只有一个根

组件的template里面的html代码只能有一个根节点



## 组件的第一种声明方式



|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>  </head>  <body>    <div id="app">      <my-component1></my-component1>    </div>    <script>     var myComponent1=Vue.extend({       template:'<div>我是一个使用Vue.extend方法创建的组件</div>'     })      Vue.component('my-component1',myComponent1);      //创建Vue实例,得到 ViewModel      var vm = new Vue({        el: '#app',        data: {},        methods: {}      });    </script>  </body>  </html> |

## 组件的第二种声明方式

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>  </head>  <body>    <div id ="app">      <my-component1></my-component1>      <my-component1></my-component1>      <my-component1></my-component1>      <my-component1></my-component1>    </div>    <script>        //注册组件      Vue.component("my-component1",{        template:'<div>不使用Vue.extend实现的组件<h1>我是h1</h1></div>'      })     //创建Vue实例,得到 ViewModel     var vm = new Vue({      el: '#app',      data: {},      methods: {}     });    </script>  </body>  </html> |

## 组件的第三种声明方式

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>  </head>  <body>    <div id ="app">      <my-component1></my-component1>      <my-component1></my-component1>      <my-component1></my-component1>      <my-component1></my-component1>    </div>    <template id="temp1">      <div>        <h1>我是HTML里面的H1</h1>        <h3>{{count}}  <button @click="add">点击</button> </h3>      </button>    </template>    <script>      var myComponent1={        template:'#temp1',        data(){  //在组件里面data是一个方法          return {            count:0          }        },        methods: {          add(){            this.count++          }        },      }      //注册组件      Vue.component("my-component1",myComponent1)     //创建Vue实例,得到 ViewModel     var vm = new Vue({      el: '#app',      data: {},      methods: {}     });    </script>  </body>  </html> |

## 掌握vue中组件的data和methods

组件中也可以有自己的Data和methods，

组件里面的data必须是一个方法 原因如下

    组件在存在是为有复用，定义一个组件之后，可能会有多个地方使用到该组件。如果按data:{}的写法，多个组件会复用同一个data，降低组件的复用性，而定义为function则不会

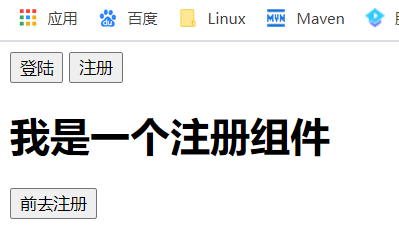
## 私有组件的使用

**上面的创建方法都是公有的组件，如果只想一个组件在某一个Vue实例里面使用，就可以使用私有组件**

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>  </head>  <body>    <div id ="app">      <my-component1></my-component1>      <my-component1></my-component1>      <my-component1></my-component1>      <my-component1></my-component1>    </div>    <div id ="app2">      11111      <my-component1></my-component1>      <my-component1></my-component1>      <my-component1></my-component1>      <my-component1></my-component1>    </div>   <template id="temp1">      <div>        <h1>我是HTML里面的H1</h1>        <h3>{{count}}  <button @click="add">点击</button> </h3>      </button>    </template>    <script>      var myComponent1={        template:'#temp1',        data(){  //在组件里面data是一个方法          return {            count:0,            name:''          }        },        methods: {          add(){            this.count++          }        },      }     //创建Vue实例,得到 ViewModel     var vm = new Vue({      el: '#app',      components:{        "myComponent1":myComponent1      },      data: {        count:0,        name:''      },      methods: {}     });     var vm2 = new Vue({      el: '#app2',      data: {        count:0,        name:''      },      methods: {}     });    </script>  </body>  </html> |

## 掌握vue组件之间的切换方式

### 使用v-if v-else v-else-if去切换组件



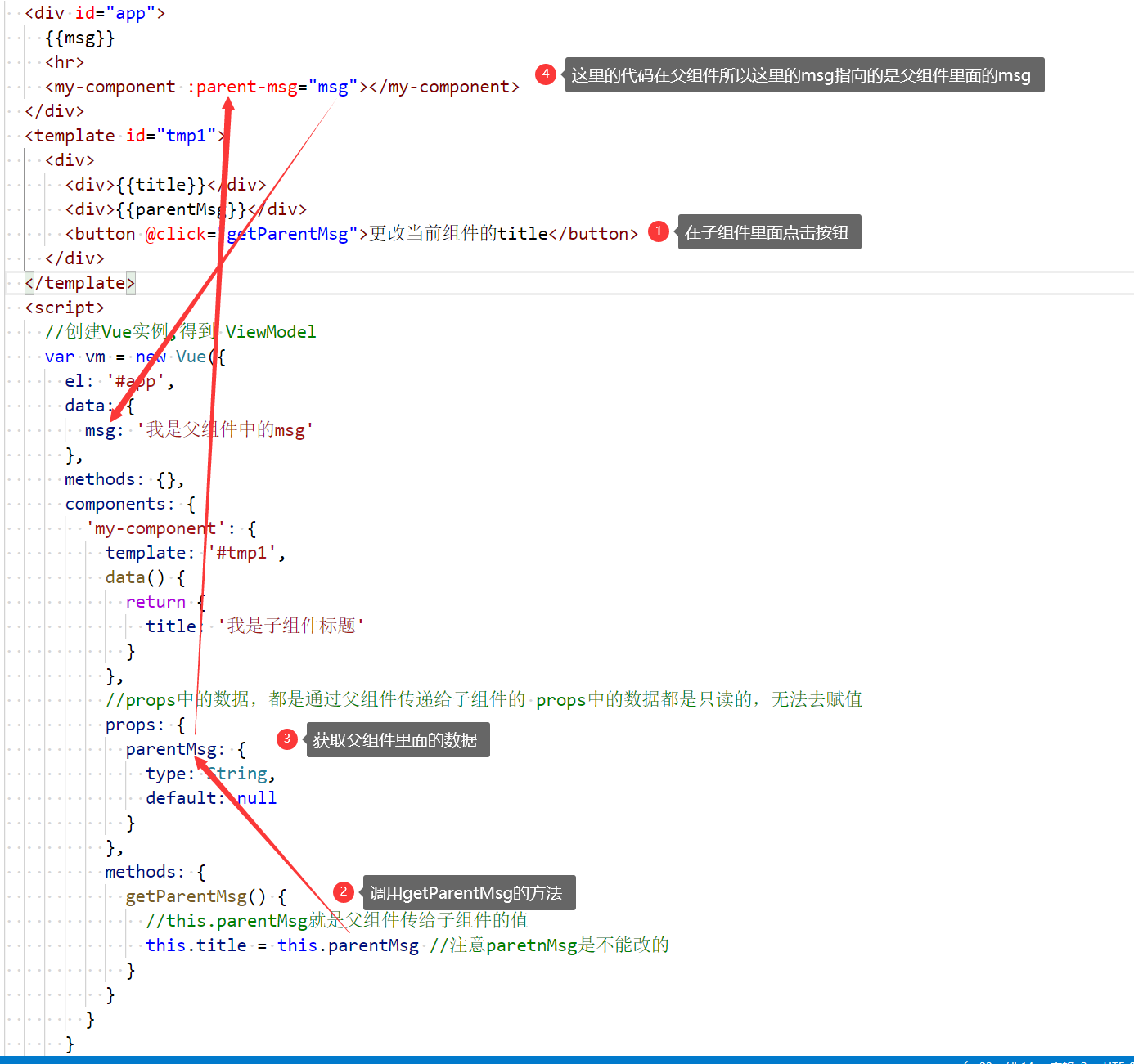
|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>  </head>  <body>    <div id ="app">      <button @click="flag=true">登陆</button>      <button  @click="flag=false">注册</button>      <div>        <login v-if="flag"></login>        <register v-else></register>      </div>    </div>    <template id="login">      <div>        <h1>我是一个登陆组件</h1>        <button>前去登陆</button>      </div>    </template>    <template id="register">      <div>        <h1>我是一个注册组件</h1>        <button>前去注册</button>      </div>    </template>    <script>      Vue.component("login",{        template:'#login',        data(){  //在组件里面data是一个方法          return {          }        },        methods: {          },      })      Vue.component("register",{        template:'#register',        data(){  //在组件里面data是一个方法          return {          }        },        methods: {          },      })     //创建Vue实例,得到 ViewModel     var vm = new Vue({      el: '#app',      data: {        flag:true,      },      methods: {}     });    </script>  </body>  </html> |

### 使用component占位 去实现[适合组件多的情况]

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>  </head>  <body>    <div id ="app">      <button @click="componentName='login'">登陆</button>      <button  @click="componentName='register'">注册</button>     <!--        Vue提供了component ，来展示对应名称的组件，这是一个占位符，        使用:is来指定展示的组件名称     -->     <component :is="componentName"></component>    </div>    <template id="login">      <div>        <h1>我是一个登陆组件</h1>        <button>前去登陆</button>      </div>    </template>    <template id="register">      <div>        <h1>我是一个注册组件</h1>        <button>前去注册</button>      </div>    </template>    <script>      Vue.component("login",{        template:'#login',        data(){  //在组件里面data是一个方法          return {          }        },        methods: {          },      })      Vue.component("register",{        template:'#register',        data(){  //在组件里面data是一个方法          return {          }        },        methods: {          },      })     //创建Vue实例,得到 ViewModel     var vm = new Vue({      el: '#app',      data: {        componentName:'login', //指定组件的名称      },      methods: {}     });    </script>  </body>  </html> |

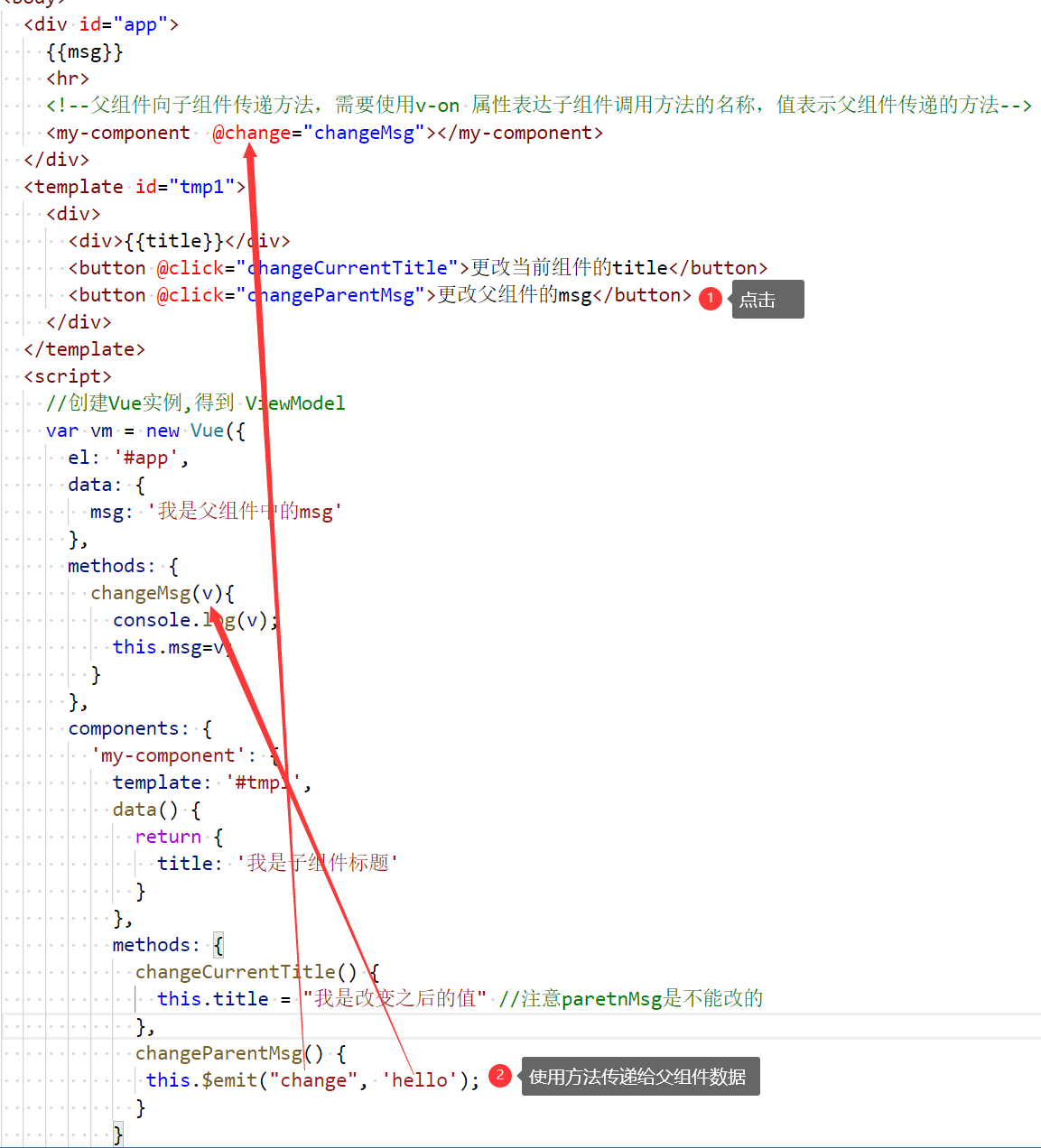
## 父组件子组件传值

### 子组件直接获取父组件数据



|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>  </head>  <body>    <div id="app">      {{msg}}      <hr>      <my-component :parent-msg="msg"></my-component>    </div>    <template id="tmp1">      <div>        <div>{{title}}</div>        <div>{{parentMsg}}</div>        <button @click="getParentMsg">更改当前组件的title</button>      </div>    </template>    <script>      //创建Vue实例,得到 ViewModel      var vm = new Vue({        el: '#app',        data: {          msg: '我是父组件中的msg'        },        methods: {},        components: {          'my-component': {            template: '#tmp1',            data() {              return {                title: '我是子组件标题'              }            },            //props中的数据，都是通过父组件传递给子组件的 props中的数据都是只读的，无法去赋值            props: {              parentMsg: {                type: String,                default: null              }            },            methods: {              getParentMsg() {                //this.parentMsg就是父组件传给子组件的值                this.title = this.parentMsg //注意paretnMsg是不能改的              }            }          }        }      });    </script>  </body>  </html> |

### 使用方法把子组里面的数据传递到父组件



|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>  </head>  <body>    <div id="app">      {{msg}}      <hr>      <!--父组件向子组件传递方法，需要使用v-on 属性表达子组件调用方法的名称，值表示父组件传递的方法-->      <my-component  @change="changeMsg"></my-component>    </div>    <template id="tmp1">      <div>        <div>{{title}}</div>        <button @click="changeCurrentTitle">更改当前组件的title</button>        <button @click="changeParentMsg">更改父组件的msg</button>      </div>    </template>    <script>      //创建Vue实例,得到 ViewModel      var vm = new Vue({        el: '#app',        data: {          msg: '我是父组件中的msg'        },        methods: {          changeMsg(v){            console.log(v);            this.msg=v;          }        },        components: {          'my-component': {            template: '#tmp1',            data() {              return {                title: '我是子组件标题'              }            },            methods: {              changeCurrentTitle() {                this.title = "我是改变之后的值" //注意paretnMsg是不能改的              },              changeParentMsg() {               this.$emit("change", 'hello');              }            }          }        }      });    </script>  </body>  </html> |

## 掌握父组件调用子组件的方法【重点掌握】



|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>  </head>  <body>    <div id="app">      <!-- <div id="div1" @click="getElement">我是内容</div> -->      <div ref="div1" @click="getElement">我是内容</div>      <div ref="div2" @click="handlerChildFunction">调用子组件的方法</div>      <my-component ref="com1"></my-component>    </div>    <template id="tmp1">      <div>      </div>    </template>    <script>      //创建Vue实例,得到 ViewModel      var vm = new Vue({        el: '#app',        methods:{          getElement(){            //1,获取dom元素，使用传递的js去获取，但是vue不推荐这种用法            //2,获取dom元素，使用ref(reference)。这个是vue提供的写法。建议使用这个            // let text=document.getElementById("div1").innerHTML;            let text=this.$refs.div1.innerText;            console.log(text);          },          handlerChildFunction(){            this.$refs.com1.childrenFunction()          }        },        components: {          'my-component': {            template: '#tmp1',            methods: {              childrenFunction(){                console.log('子组件的方法被调用了')              }            }          }        }      });    </script>  </body>  </html> |

# 【掌握】路由

## 什么是路由及作用

什么是路由

后端：对于普通网站同，所有的**超链接**都对应一个url地址，每个url都对应服务器上的资源

前端：对于单页面应用程序来说的，主要通过url中的#（hash）去实现的，(当关于a标签)

**作用：就是用来做跳转的**

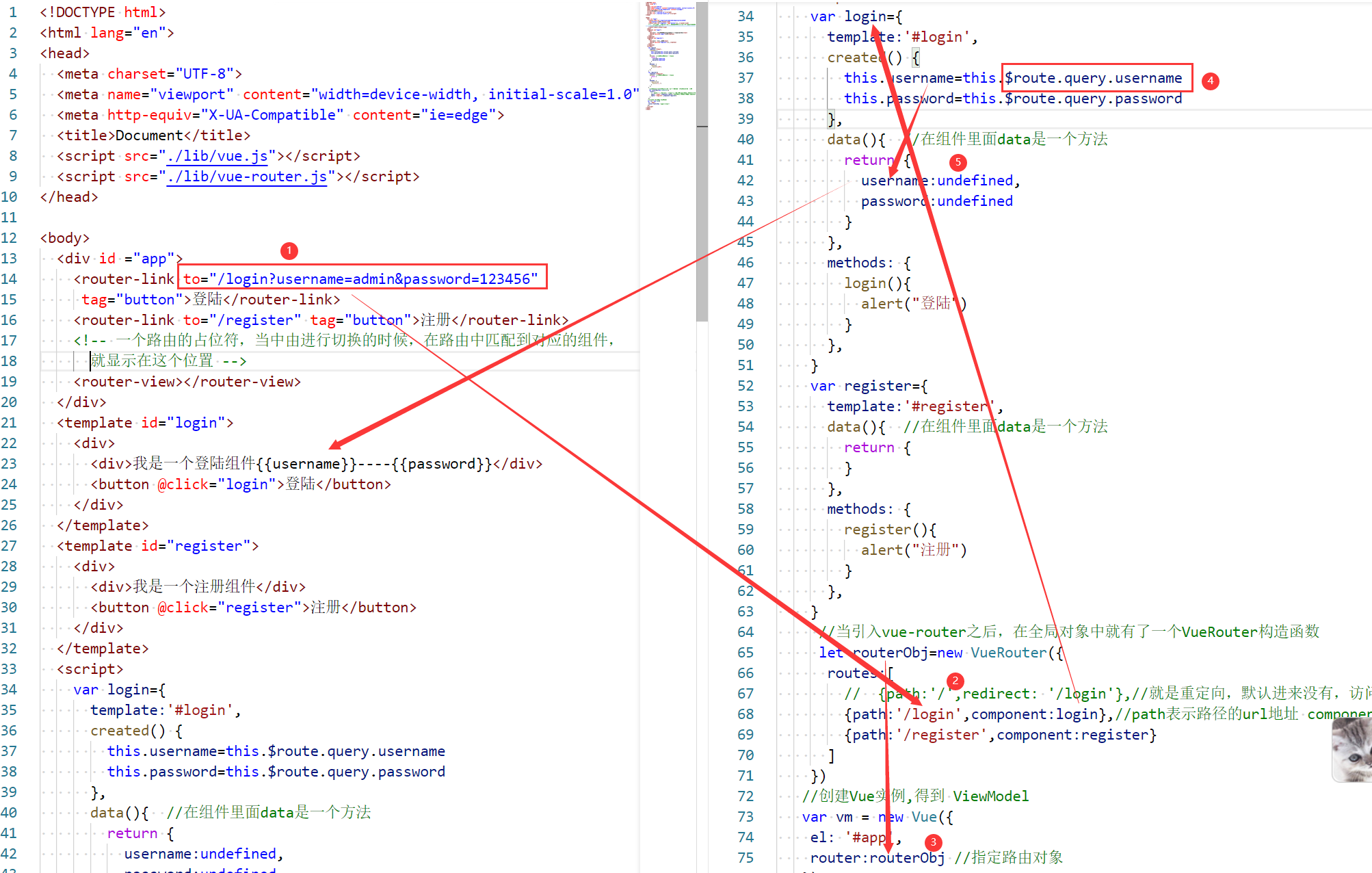
## 掌握vue路由的基本使用

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>    <script src="./lib/vue-router.js"></script>  </head>  <body>    <div id="app">      <!-- 默认渲染成a标签 可以使用tag来改变渲染的标签 -->      <router-link to="/login" tag="button">登陆</router-link>      <router-link to="/register" tag="button">注册</router-link>        <!--一个路由的占位符，当中由进行切换的时候，在路由中匹配到对应的组件，就显示在这个位置 -->      <router-view></router-view>    </div>    <template id="login">      <div>        <div>我是一个登陆组件</div>        <button @click="login">登陆</button>      </div>    </template>    <template id="register">      <div>        <div>我是一个注册组件</div>        <button @click="register">注册</button>      </div>    </template>    <script>        var login={        template: '#login',        methods:{          login(){           alert('登陆');          }        }      }      var register={        template: '#register',        methods:{          register(){           alert('注册');          }        }      }      //当引入vue-router之后，在全局对象中就有了一个VueRouter构造函数      let routerObj=new VueRouter({        routes:[          {path:'/',redirect: '/login'},//就是重定向，默认进来没有，访问/路径时，会被重定向到指定的路由路径，非请求的重定向          {path:'/login',component:login},//path表示路径的url地址 component:代表路由要跳转的组件          {path:'/register',component:register}        ]      })      //创建Vue实例,得到 ViewModel      var vm = new Vue({        el: '#app',        data: {},        router:routerObj //指定路由对象      });      </script>  </body>  </html> |

127.0.0.1:8080/user?username=admin&password=123456

## 掌握路由跳转传参的方式

### 地址传参【不推荐】数据在$route.query里面



|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>    <script src="./lib/vue-router.js"></script>  </head>  <body>    <div id ="app">      <router-link to="/login?username=admin&password=123456"       tag="button">登陆</router-link>      <router-link to="/register" tag="button">注册</router-link>      <!-- 一个路由的占位符，当中由进行切换的时候，在路由中匹配到对应的组件，        就显示在这个位置 -->      <router-view></router-view>    </div>    <template id="login">      <div>        <div>我是一个登陆组件{{username}}----{{password}}</div>        <button @click="login">登陆</button>      </div>    </template>    <template id="register">      <div>        <div>我是一个注册组件</div>        <button @click="register">注册</button>      </div>    </template>    <script>      var login={        template:'#login',        created() {          this.username=this.$route.query.username          this.password=this.$route.query.password        },        data(){  //在组件里面data是一个方法          return {            username:undefined,            password:undefined          }        },        methods: {          login(){            alert("登陆")          }        },      }      var register={        template:'#register',        data(){  //在组件里面data是一个方法          return {          }        },        methods: {          register(){            alert("注册")          }        },      }       //当引入vue-router之后，在全局对象中就有了一个VueRouter构造函数       let routerObj=new VueRouter({        routes:[          //  {path:'/',redirect: '/login'},//就是重定向，默认进来没有，访问/路径时，会被重定向到指定的路由路径，非请求的重定向          {path:'/login',component:login},//path表示路径的url地址 component:代表路由要跳转的组件          {path:'/register',component:register}        ]      })     //创建Vue实例,得到 ViewModel     var vm = new Vue({      el: '#app',      router:routerObj //指定路由对象     });    </script>  </body>  </html> |

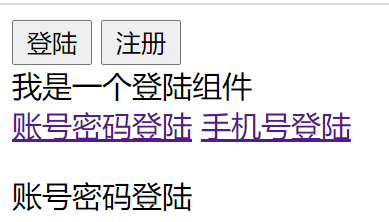
### rest风格【推荐】数据在$route.param里面



|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>    <script src="./lib/vue-router.js"></script>  </head>  <body>    <div id ="app">      <router-link to="/login?username=admin&password=123456"       tag="button">登陆</router-link>      <router-link to="/register/zhangsan/123456" tag="button">注册</router-link>      <!-- 一个路由的占位符，当中由进行切换的时候，在路由中匹配到对应的组件，就显示在这个位置 -->      <router-view></router-view>    </div>    <template id="login">      <div>        <div>我是一个登陆组件{{username}}----{{password}}</div>        <button @click="login">登陆</button>      </div>    </template>    <template id="register">      <div>        <div>我是一个注册组件{{username}}----{{password}}</div>        <button @click="register">注册</button>      </div>    </template>    <script>      var login={        template:'#login',        created() {          this.username=this.$route.query.username          this.password=this.$route.query.password          console.log(this.$route.query.username+"  "+this.$route.query.password)        },        data(){  //在组件里面data是一个方法          return {            username:undefined,            password:undefined          }        },        methods: {          login(){            alert("登陆")          }        },      }      var register={        template:'#register',        created() {          console.log(this.$route)          this.username=this.$route.params.username          this.password=this.$route.params.password        },        data(){  //在组件里面data是一个方法          return {            username:undefined,            password:undefined          }        },        methods: {          register(){            alert("注册")          }        },      }       //当引入vue-router之后，在全局对象中就有了一个VueRouter构造函数       let routerObj=new VueRouter({        routes:[          //  {path:'/',redirect: '/login'},//就是重定向，默认进来没有，访问/路径时，会被重定向到指定的路由路径，非请求的重定向          {path:'/login',component:login},//path表示路径的url地址 component:代表路由要跳转的组件          {path:'/register/:username/:password',component:register}        ]      })     //创建Vue实例,得到 ViewModel     var vm = new Vue({      el: '#app',      router:routerObj //指定路由对象     });    </script>  </body>  </html> |

## 掌握路由的嵌套使用

就是路由里面套路由，如登陆页面可以使用手机或者账号密码登陆



|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>    <script src="./lib/vue-router.js"></script>  </head>  <body>    <div id="app">      <!-- 默认渲染成a标签 可以使用tag来改变渲染的标签 -->      <router-link to="/login?username=admin&password=123456" tag="button">登陆</router-link>      <router-link to="/register/zhangsan/123456" tag="button">注册</router-link>        <!--一个路由的占位符，当中由进行切换的时候，在路由中匹配到对应的组件，就显示在这个位置 -->      <router-view></router-view>    </div>    <template id="login">      <div>        <div>我是一个登陆组件</div>        <router-link to="/login/loginByPassword">账号密码登陆</router-link>        <router-link to="/login/loginBySms">手机号登陆</router-link>        <router-view></router-view>      </div>    </template>    <template id="register">      <div>        <div>我是一个注册组件</div>      </div>    </template>    <script>        var login={        template: '#login',      }      var loginByPassword={        template: '<p>账号密码登陆</p>',      }      var loginBySms={        template: '<p>手机号登陆</p>',      }      var register={        template: '#register',      }      //当引入vue-router之后，在全局对象中就有了一个VueRouter构造函数      let routerObj=new VueRouter({        routes:[          {path:'/',redirect: '/login'},          {            path:'/login',component:login,            children:[              {path:'',redirect: 'loginByPassword'},              {path:'loginByPassword',component:loginByPassword},              {path:'loginBySms',component:loginBySms}            ]         },//path表示路径的url地址 component:代表路由要跳转的组件          {path:'/register/:username/:password',component:register}        ]      })      //创建Vue实例,得到 ViewModel      var vm = new Vue({        el: '#app',        data: {},        router:routerObj //指定路由对象      });      </script>  </body>  </html> |

## 掌握编程式导航的路由跳转

什么是编程试导航

就是通过按钮点击调用方法通过方法去转到某个路由

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="./lib/vue.js"></script>    <script src="./lib/vue-router.js"></script>  </head>  <body>    <div id ="app">      <button @click="toLogin">登陆</button>      <button @click="toRegister">注册</button>      <!-- 一个路由的占位符，当中由进行切换的时候，在路由中匹配到对应的组件，就显示在这个位置 -->      <router-view></router-view>    </div>    <template id="login">      <div>        <div>我是一个登陆组件</div>      </div>    </template>    <template id="register">      <div>        <div>我是一个注册组件</div>      </div>    </template>    <script>      var login={        template:'#login',        created() {          console.log('我是登陆组件里面的')          console.log(this.$route.query)        },      }      var register={        template:'#register',      }       //当引入vue-router之后，在全局对象中就有了一个VueRouter构造函数       let routerObj=new VueRouter({        routes:[          // {path:'/',redirect: '/login'},//就是重定向，默认进来没有，访问/路径时，会被重定向到指定的路由路径，非请求的重定向          {path:'/login',component:login},//path表示路径的url地址 component:代表路由要跳转的组件          {path:'/register',component:register}        ]      })     //创建Vue实例,得到 ViewModel     var vm = new Vue({      el: '#app',      methods: {        toLogin(){          this.$router.push({'path':'/login',query:{username:'admin'}})//跳转并传参        },        toRegister(){          this.$router.push('/register')        }      },      router:routerObj //指定路由对象     });    </script>  </body>  </html> |

# 【掌握】node和npm的说明

## 什么是node和npm

Node：是JavaScript的运行时环境。Node并不是一个新的语言，只是js的一个环境，同样的比如：tomcat。Node为js提供了更强大的操作方式，如：在浏览器中，js是无法操作文件的，而node提供了文件操作。在浏览器中，js无法写服务接口，node提供了后端代码编写的功能（写后台、操作数据库）。尽管如此，Node依然是个前端技术，并不会真正的使用node去写后端代码，node更多的是给前端项目做配置，如：跨域代理。

Npm：npm是node提供的一个包管理工具，类似于maven。通过npm去安装依赖包，就不需要在页面上使用script标签引入了。

命令：-g全局安装，任何项目都可以用。--save-dev开发环境安装，不会打包到生产。生产安装（默认）

## node的下载

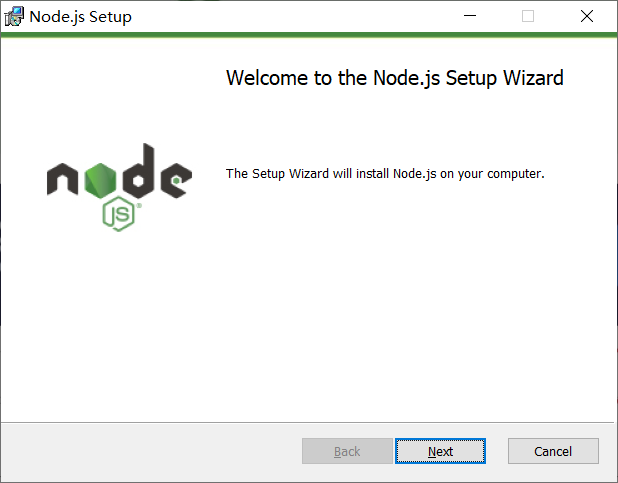
<http://nodejs.cn/>

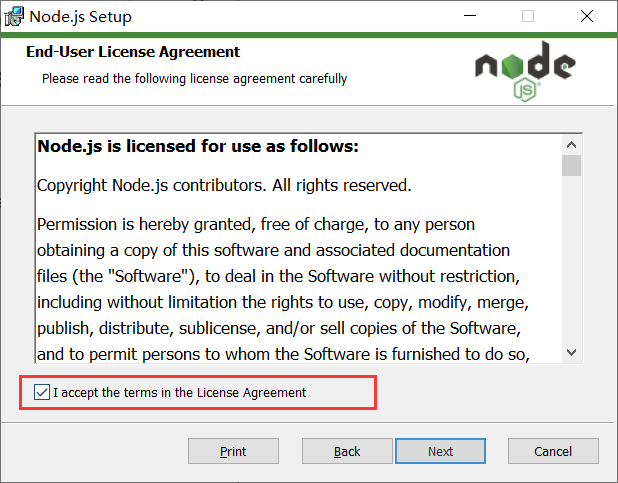


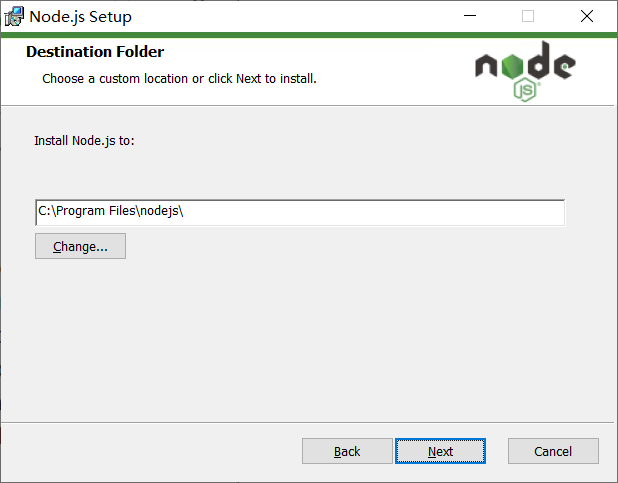


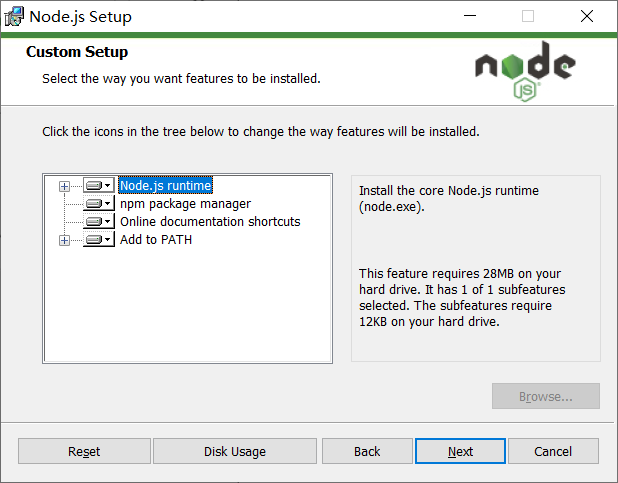
IMG_258

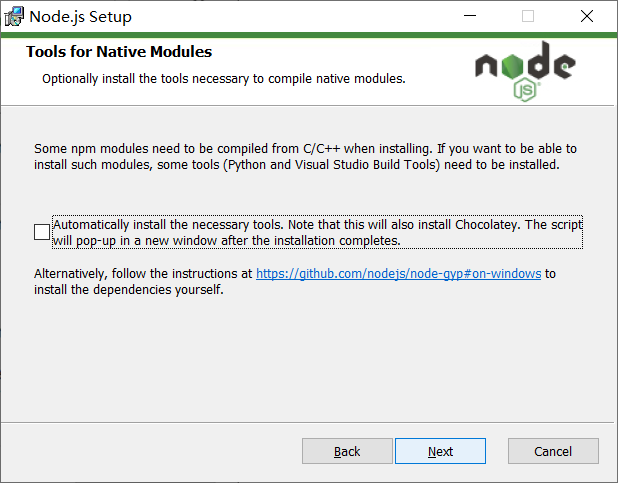
## Node的安装

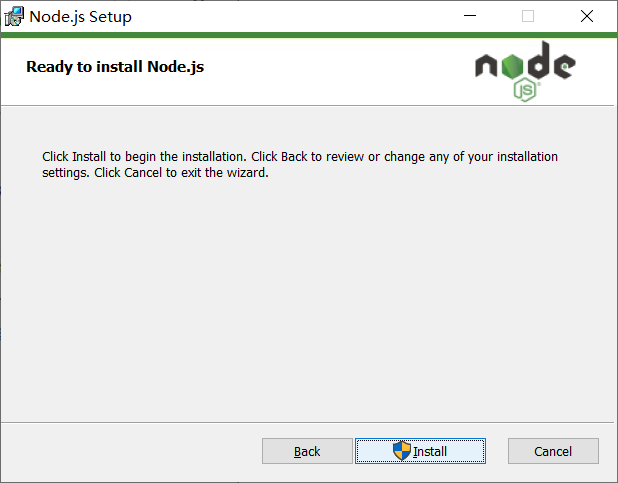


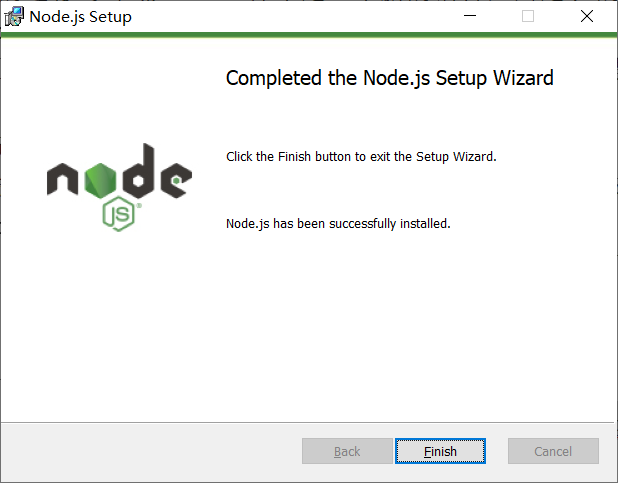






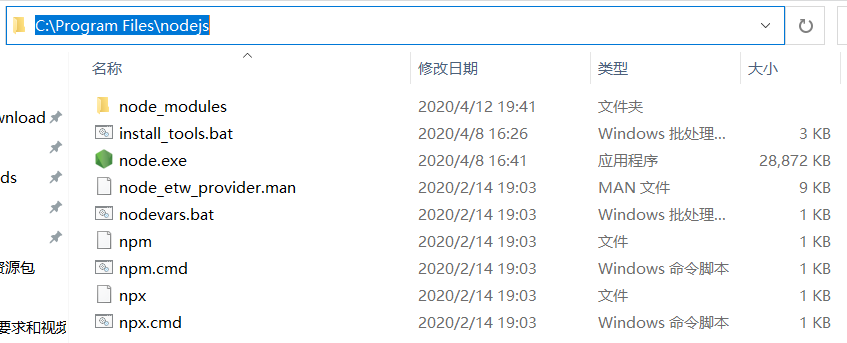






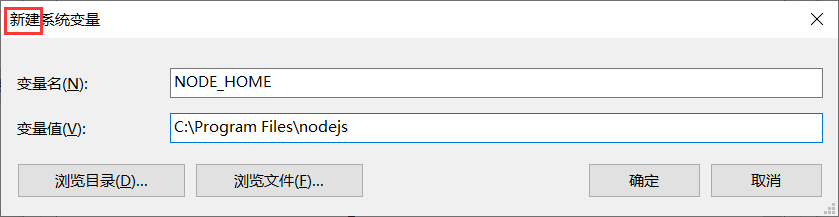
## 配置node的环境变量

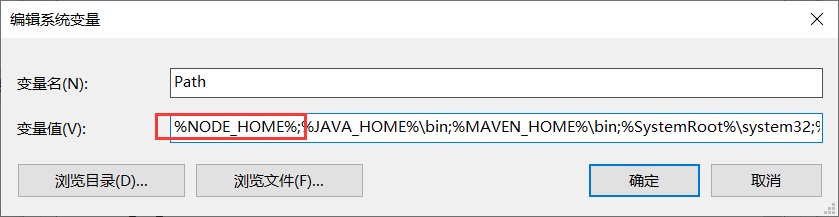
### 配置









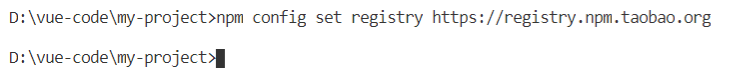


### 测试环境变量



## npm镜像加速配置

npm config set registry https://registry.npm.taobao.org



# 【掌握】熟悉vue-cli项目脚手架

## 什么是vue-cli

vue脚手架指的是vue-cli，它是一个专门为单页面应用快速搭建繁杂的脚手架，它可以轻松的创建新的应用程序而且可用于自动生成vue和webpack的项目模板。

vue-cli是有Vue提供的一个官方cli，专门为单页面应用快速搭建繁杂的脚手架。它是用于自动生成vue.js+webpack的项目模板。

## vue-cli安装

### 全局安装 npm install -g @vue/cli@3.7.0

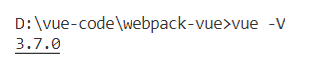
进入工作空间根目录



执行命令

npm install -g @vue/cli@3.7.0

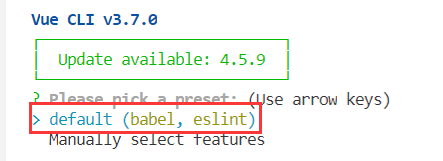
### **查看版本/是否安装成功 vue -V**



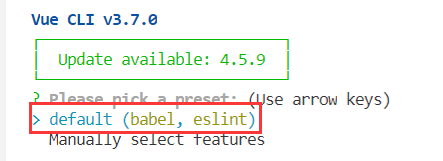
## **使用vue-cli创建项目-命令【了解】**

### **默认和手动创建项目**

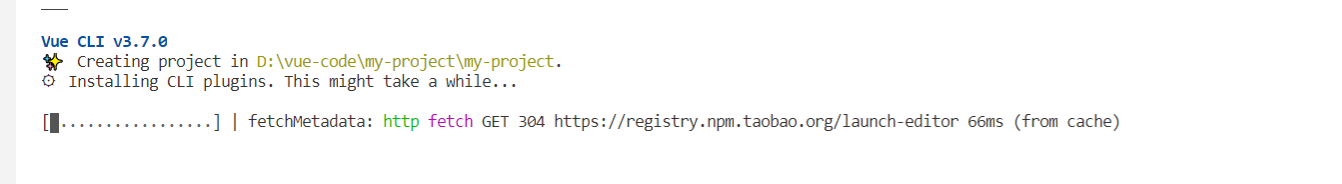
vue create my-project



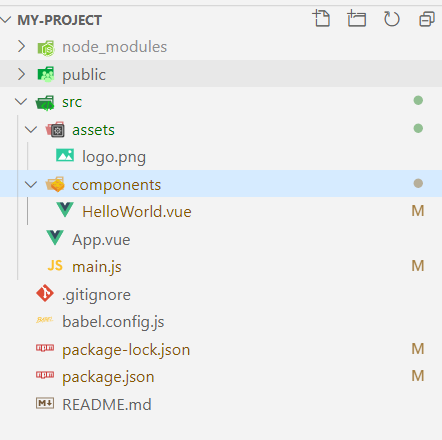
### **手动创建项目后选择安装功能，上下键移动，空格选中/取消**



创建中



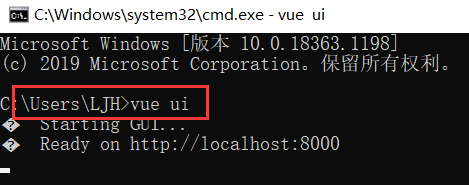
### 创建成功使用vscode打开



以上的为自动模式，手动模式看下面的博客<https://www.jianshu.com/p/06a9f112867d>

## 使用vue-cli创建项目-命令【重点掌握】

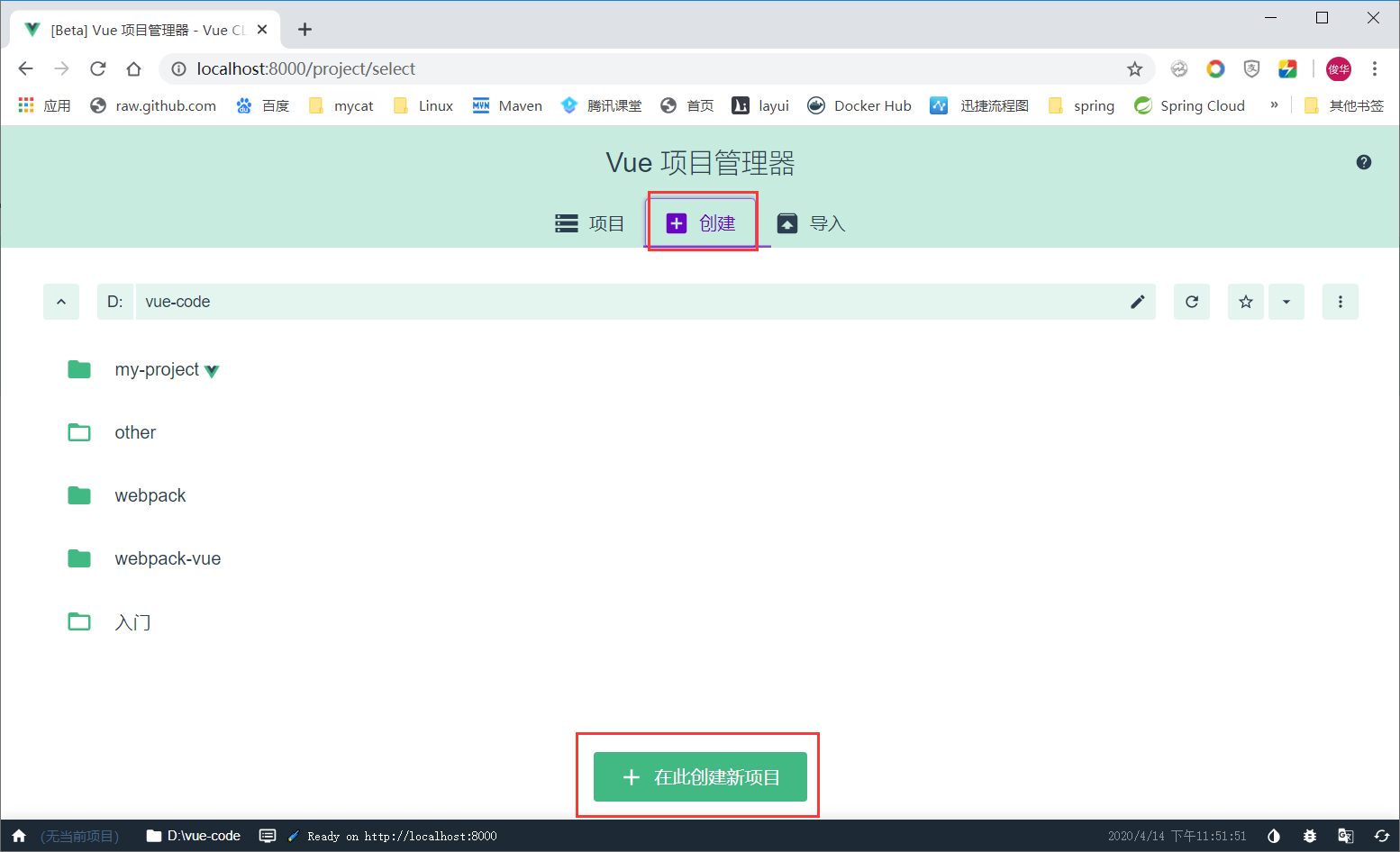
### 命令启动图型化界面



### 界面效果



### 界面选择创建



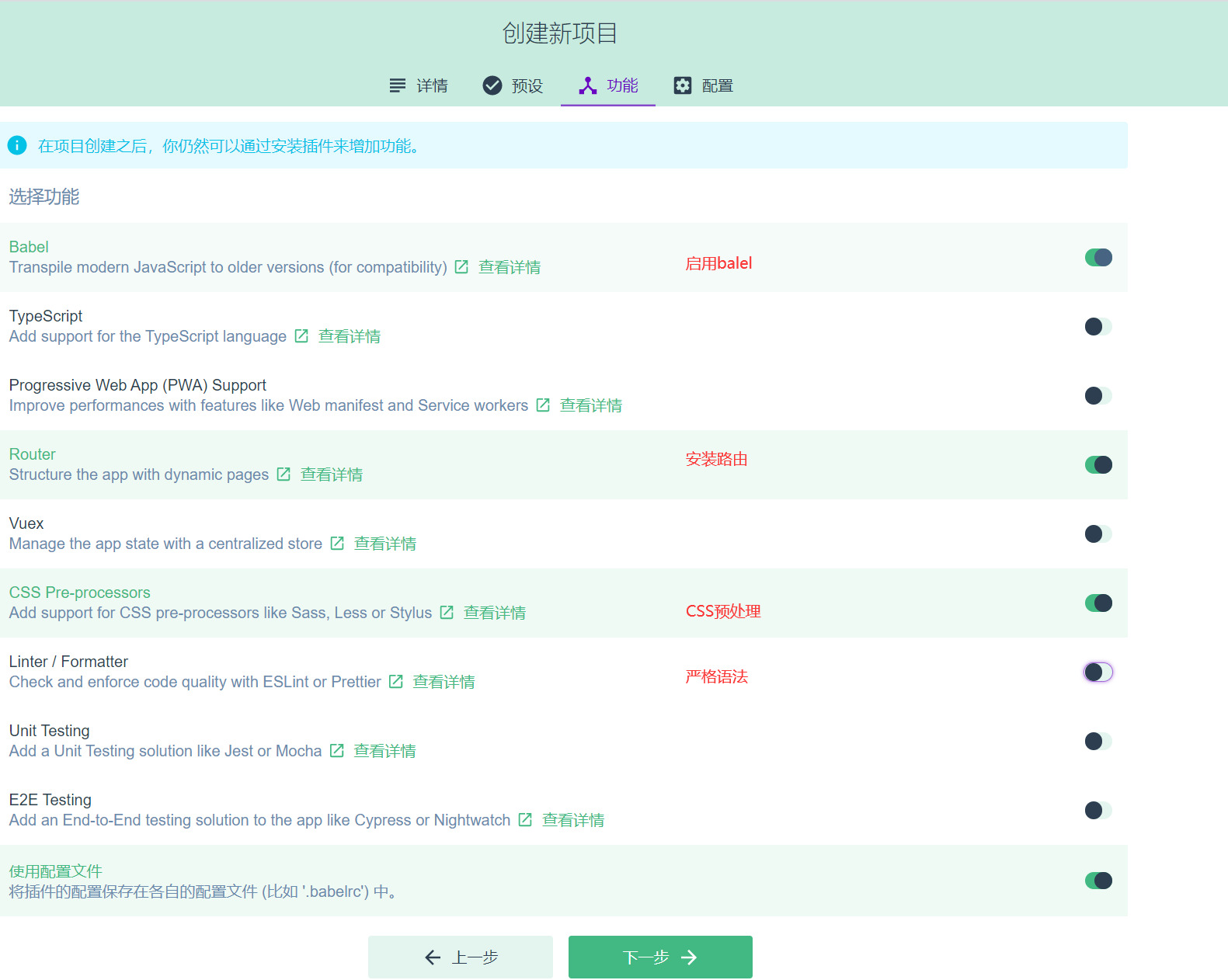
### 输入相关配置



### 下一步预设



### 下一步选择安装



### 下一步





## 项目仪表盘

### 仪表盘



### 安装axios的依赖

这个东东是用来发网络请求的





### 使用vscode打开项目

# 【掌握】export导出成员精讲

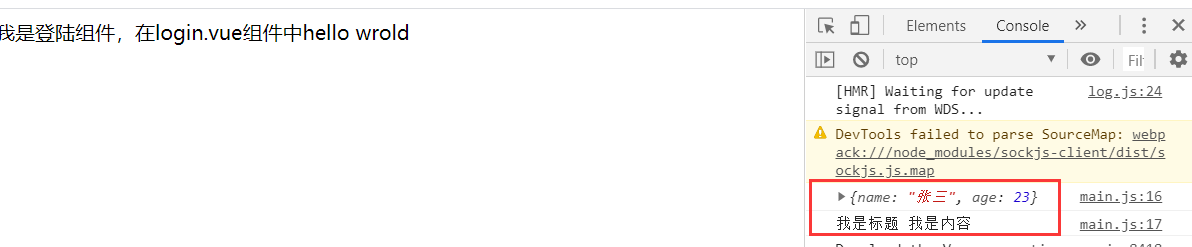
## 建test.js来说明

|  |
| --- |
| // 在webpack.config.js里面有module.export  //这个是Node中向外暴露成员的方式  // module.exports = {}  // 在ES6中，也通过规范的形式，规定了ES6如何导入和导出模块  // ES6中，使用 import 模块名称 from '模块标识符' import '模块路径'两种方式导入模块  // ES6中,使用export default 和export向外暴露成员.  var info = {    name: '张三',    age: 23  }  export default info  // 注意：export default 向外暴露的成员，可以用任意的变量接收  // 注意：在一个模块中，export default只允许向外暴露一次  // 注意: 在一个模块中, 可以同时使用export default和export  export var title = '我是标题'  export var content = '我是内容'  // 注意：使用export向外暴露成员，只能使用{ }的形式去接收，叫做【按需引入】，按需引入是开发中非常常用的方式  // 注意：export可以向外暴露多个成员，同时，如果某些成员，我们在import的时候，不需要，则可以不在 { } 中定义  // 注意：使用export导出的成员，必须严格按照export导出时候的名称，来使用 { }接收  // 注意：使用export导出的成员，如果就想换个名字接收，可以使用as 起个别名  // Node中，引入使用的是var 名称 = require('模块标识符') |

## 修改main.js

|  |
| --- |
| import obj ,{title,content} from './test'  console.log(obj)  console.log(title)  console.log(content) |

## 测试



# 【掌握】ui框架

## 市面常用ui框架说明

## ui框架 PC端

* bootstrap：比较强大的一款前端ui框架，存在的时间也比较久，但是依赖于jQuery，不符合vue的开发思想，不建议用
* layui：经典的模块化前端框架，虽然使用时不需要引入jQuery，但是底层是依赖于jQuery的，并且非常依赖于dom操作，不建议使用
* iview：一套基于 Vue.js 的高质量 UI 组件库。对vue兼容性极强，功能强大，组件丰富，推荐。
* ant-design-vue：这是蚂蚁金服的开源ui框架。大名鼎鼎的jeecg就是使用的这个，完美兼容vue
* elementui：目前vue开发者使用最多的ui框架，饿了么团队开发（阿里），完美兼容vue

## ui框架 移动端

* mui：DCloud出品的ui框架，可以实现半原生开发。曾经依赖于jQuery
* mintui：饿了么团队出品的移动端ui框架，兼容vue，但是不太关注，功能较少
* antui：蚂蚁金服移动端ui框架，完美兼容vue，封装不太好
* vant：有赞出品的前端ui框架，完美兼容vue

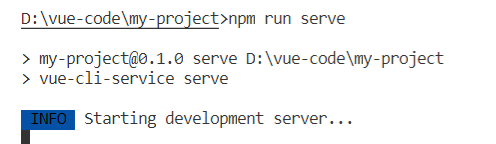
# 【掌握】elementui的简单使用

## 接上一个用vue-cli创建好的项目

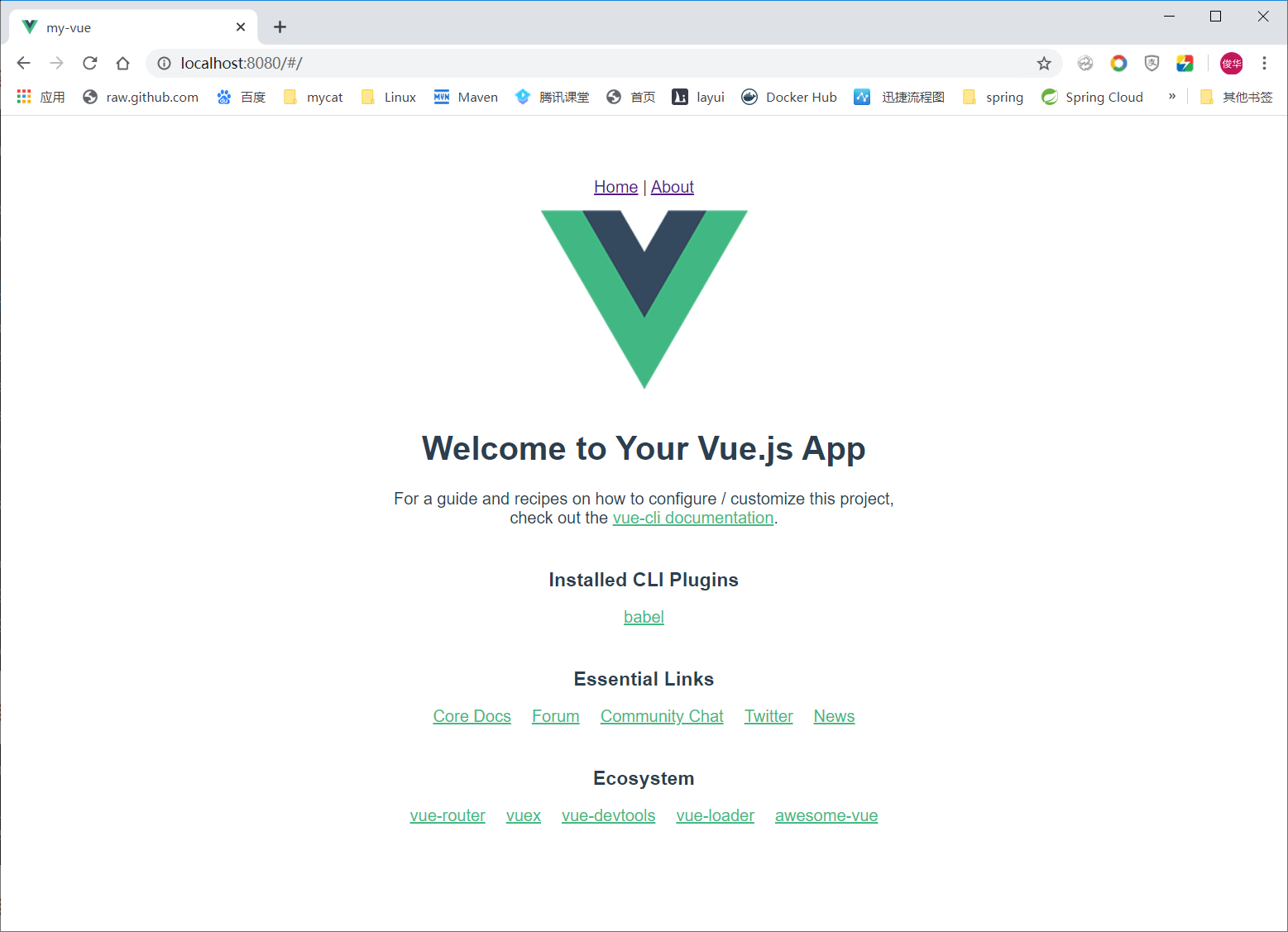
### 项目结构



### 启动项目



### 访问项目



## 打开element-ui官网开始搞事情

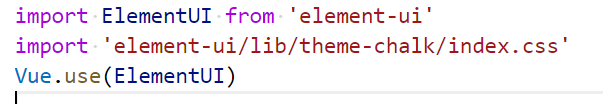
[https://element.eleme.cn/#/zh-CN/component/installation](https://element.eleme.cn/" \l "/zh-CN/component/installation)

### 安装element-ui

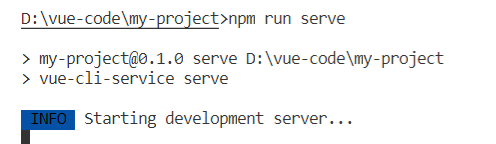


npm i element-ui -S

#### 引入element-ui



### 启动项目



npm run serve

## 树型组件

### 创建view/Tree.vue

|  |
| --- |
| <template>    <div>      <el-tree        :data="data"        show-checkbox        default-expand-all        node-key="id"        ref="tree"        highlight-current        :props="defaultProps"      ></el-tree>      <div class="buttons">        <el-button @click="getCheckedNodes">通过 node 获取</el-button>        <el-button @click="getCheckedKeys">通过 key 获取</el-button>        <el-button @click="setCheckedNodes">通过 node 设置</el-button>        <el-button @click="setCheckedKeys">通过 key 设置</el-button>        <el-button @click="resetChecked">清空</el-button>      </div>    </div>  </template>  <script>    export default {      methods: {        getCheckedNodes() {          console.log(this.$refs.tree.getCheckedNodes());        },        getCheckedKeys() {          console.log(this.$refs.tree.getCheckedKeys());        },        setCheckedNodes() {          this.$refs.tree.setCheckedNodes([{            id: 5,            label: '二级 2-1'          }, {            id: 9,            label: '三级 1-1-1'          }]);        },        setCheckedKeys() {          this.$refs.tree.setCheckedKeys([3]);        },        resetChecked() {          this.$refs.tree.setCheckedKeys([]);        }      },      data() {        return {          data: [{            id: 1,            label: '一级 1',            children: [{              id: 4,              label: '二级 1-1',              children: [{                id: 9,                label: '三级 1-1-1'              }, {                id: 10,                label: '三级 1-1-2'              }]            }]          }, {            id: 2,            label: '一级 2',            children: [{              id: 5,              label: '二级 2-1'            }, {              id: 6,              label: '二级 2-2'            }]          }, {            id: 3,            label: '一级 3',            children: [{              id: 7,              label: '二级 3-1'            }, {              id: 8,              label: '二级 3-2'            }]          }],          defaultProps: {            children: 'children',            label: 'label'          }        };      }    };  </script>  </script> |

### 修改router/index.js



### 修改App.vue

|  |
| --- |
|  |

### 效果



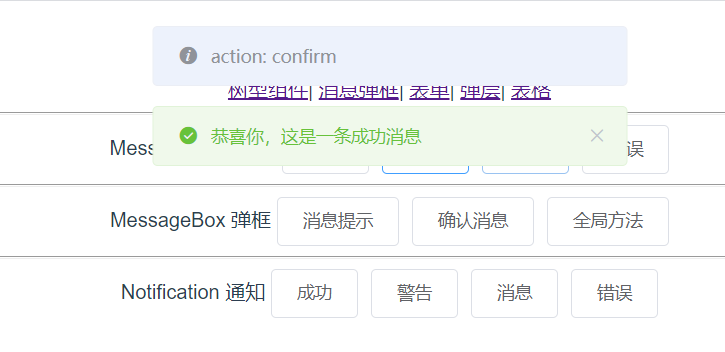
## 消息弹框

### 创建view/Notice.vue

|  |
| --- |
| <template>    <div>      <div>        Message 消息提示        <el-button :plain="true" @click="open1">消息</el-button>        <el-button :plain="true" @click="open2">成功</el-button>        <el-button :plain="true" @click="open3">警告</el-button>        <el-button :plain="true" @click="open4">错误</el-button>      </div>      <hr />      <div>        MessageBox 弹框        <el-button :plain="true" @click="open5">消息提示</el-button>        <el-button :plain="true" @click="open6">确认消息</el-button>        <el-button :plain="true" @click="open7">全局方法</el-button>      </div>       <hr />      <div>        Notification 通知        <el-button plain @click="open8">成功</el-button>        <el-button plain @click="open9">警告</el-button>        <el-button plain @click="open10">消息</el-button>        <el-button plain @click="open11">错误</el-button>      </div>    </div>  </template>  <script>  export default {    methods: {      open1() {        this.$message({          showClose: true,          message: "这是一条消息提示"        });      },      open2() {        this.$message({          showClose: true,          message: "恭喜你，这是一条成功消息",          type: "success"        });      },      open3() {        this.$message({          showClose: true,          message: "警告哦，这是一条警告消息",          type: "warning"        });      },      open4() {        this.$message({          showClose: true,          message: "错了哦，这是一条错误消息",          type: "error"        });      },      open5() {        this.$alert("这是一段内容", "标题名称", {          confirmButtonText: "确定",          callback: action => {            this.$message({              type: "info",              message: `action: ${action}`            });          }        });      },      open6() {        this.$confirm("此操作将永久删除该文件, 是否继续?", "提示", {          confirmButtonText: "确定",          cancelButtonText: "取消",          type: "warning"        })          .then(() => {            this.$message({              type: "success",              message: "删除成功!"            });          })          .catch(() => {            this.$message({              type: "info",              message: "已取消删除"            });          });      },      open7() {        this.$confirm("确定", "标题");      },      open8() {        this.$notify({          title: "成功",          message: "这是一条成功的提示消息",          type: "success"        });      },      open9() {        this.$notify({          title: "警告",          message: "这是一条警告的提示消息",          type: "warning"        });      },      open10() {        this.$notify.info({          title: "消息",          message: "这是一条消息的提示消息"        });      },      open11() {        this.$notify.error({          title: "错误",          message: "这是一条错误的提示消息"        });      }    }  };  </script> |

### 修改router/index.js

### 效果



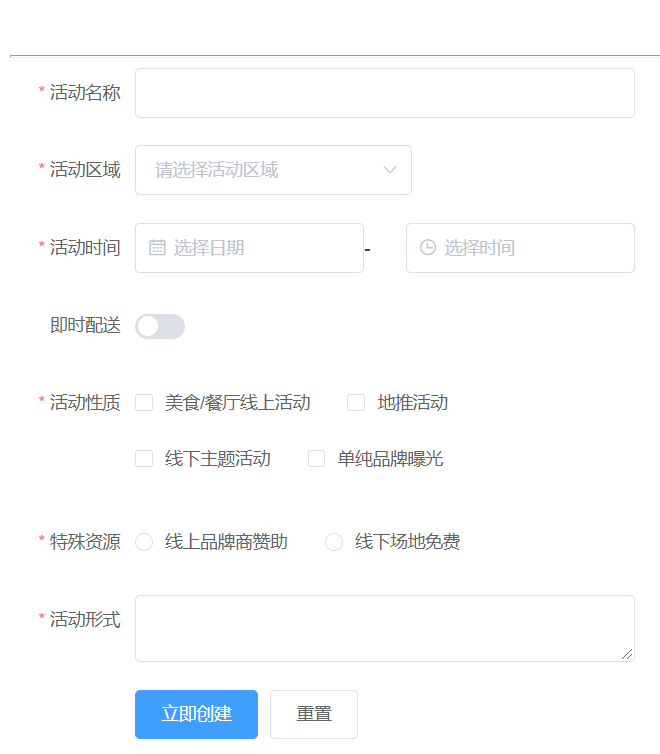
## 表单

### 创建view/Form.vue

|  |
| --- |
| <template>    <div style="width:500px;text-align:left">      <el-form        :model="ruleForm"        :rules="rules"        ref="ruleForm"        label-width="100px"        class="demo-ruleForm"      >        <el-form-item label="活动名称" prop="name">          <el-input v-model="ruleForm.name"></el-input>        </el-form-item>        <el-form-item label="活动区域" prop="region">          <el-select v-model="ruleForm.region" placeholder="请选择活动区域">            <el-option label="区域一" value="shanghai"></el-option>            <el-option label="区域二" value="beijing"></el-option>          </el-select>        </el-form-item>        <el-form-item label="活动时间" required>          <el-col :span="11">            <el-form-item prop="date1">              <el-date-picker                type="date"                placeholder="选择日期"                v-model="ruleForm.date1"                style="width: 100%;"              ></el-date-picker>            </el-form-item>          </el-col>          <el-col class="line" :span="2">-</el-col>          <el-col :span="11">            <el-form-item prop="date2">              <el-time-picker placeholder="选择时间" v-model="ruleForm.date2" style="width: 100%;"></el-time-picker>            </el-form-item>          </el-col>        </el-form-item>        <el-form-item label="即时配送" prop="delivery">          <el-switch v-model="ruleForm.delivery"></el-switch>        </el-form-item>        <el-form-item label="活动性质" prop="type">          <el-checkbox-group v-model="ruleForm.type">            <el-checkbox label="美食/餐厅线上活动" name="type"></el-checkbox>            <el-checkbox label="地推活动" name="type"></el-checkbox>            <el-checkbox label="线下主题活动" name="type"></el-checkbox>            <el-checkbox label="单纯品牌曝光" name="type"></el-checkbox>          </el-checkbox-group>        </el-form-item>        <el-form-item label="特殊资源" prop="resource">          <el-radio-group v-model="ruleForm.resource">            <el-radio label="线上品牌商赞助"></el-radio>            <el-radio label="线下场地免费"></el-radio>          </el-radio-group>        </el-form-item>        <el-form-item label="活动形式" prop="desc">          <el-input type="textarea" v-model="ruleForm.desc"></el-input>        </el-form-item>        <el-form-item>          <el-button type="primary" @click="submitForm('ruleForm')">立即创建</el-button>          <el-button @click="resetForm('ruleForm')">重置</el-button>        </el-form-item>      </el-form>    </div>  </template>  <script>  export default {    data() {      return {        ruleForm: {          name: "",          region: "",          date1: "",          date2: "",          delivery: false,          type: [],          resource: "",          desc: ""        },        rules: {          name: [            { required: true, message: "请输入活动名称", trigger: "blur" },            { min: 3, max: 5, message: "长度在 3 到 5 个字符", trigger: "blur" }          ],          region: [            { required: true, message: "请选择活动区域", trigger: "change" }          ],          date1: [            {              type: "date",              required: true,              message: "请选择日期",              trigger: "change"            }          ],          date2: [            {              type: "date",              required: true,              message: "请选择时间",              trigger: "change"            }          ],          type: [            {              type: "array",              required: true,              message: "请至少选择一个活动性质",              trigger: "change"            }          ],          resource: [            { required: true, message: "请选择活动资源", trigger: "change" }          ],          desc: [{ required: true, message: "请填写活动形式", trigger: "blur" }]        }      };    },    methods: {      submitForm(formName) {        this.$refs[formName].validate(valid => {          if (valid) {            alert("submit!");          } else {            console.log("error submit!!");            return false;          }        });      },      resetForm(formName) {        this.$refs[formName].resetFields();      }    }  };  </script> |

### 修改router/index.js

### 效果



## 弹层

### 创建view/Dialog.vue

|  |
| --- |
| <template>    <div>      <el-button :plain="true" @click="open1">基本用法</el-button>      <el-button :plain="true" @click="open2">自定义内容</el-button>      <el-button :plain="true" @click="open3">嵌套的 Dialog</el-button>      <el-dialog title="提示" :visible.sync="dialogVisible" width="30%" :before-close="handleClose">        <span>这是一段信息</span>        <span slot="footer" class="dialog-footer">          <el-button @click="dialogVisible = false">取 消</el-button>          <el-button type="primary" @click="dialogVisible = false">确 定</el-button>        </span>      </el-dialog>      <el-dialog title="收货地址" :visible.sync="dialogFormVisible">        <el-form :model="form">          <el-form-item label="活动名称" :label-width="formLabelWidth">            <el-input v-model="form.name" autocomplete="off"></el-input>          </el-form-item>          <el-form-item label="活动区域" :label-width="formLabelWidth">            <el-select v-model="form.region" placeholder="请选择活动区域">              <el-option label="区域一" value="shanghai"></el-option>              <el-option label="区域二" value="beijing"></el-option>            </el-select>          </el-form-item>        </el-form>        <div slot="footer" class="dialog-footer">          <el-button @click="dialogFormVisible = false">取 消</el-button>          <el-button type="primary" @click="dialogFormVisible = false">确 定</el-button>        </div>      </el-dialog>      <el-dialog title="外层 Dialog" :visible.sync="outerVisible">        <el-dialog width="30%" title="内层 Dialog" :visible.sync="innerVisible" append-to-body></el-dialog>        <div slot="footer" class="dialog-footer">          <el-button @click="outerVisible = false">取 消</el-button>          <el-button type="primary" @click="innerVisible = true">打开内层 Dialog</el-button>        </div>      </el-dialog>    </div>  </template>  <script>  export default {    data() {      return {        dialogVisible: false,        dialogFormVisible: false,        formLabelWidth: "120px",        outerVisible: false,        innerVisible: false,        form: {          name: "",          region: "",          date1: "",          date2: "",          delivery: false,          type: [],          resource: "",          desc: ""        }      };    },    methods: {      open1() {        this.dialogVisible = true;      },      open2() {        this.dialogFormVisible = true;      },      open3() {        this.outerVisible = true;      },      handleClose(done) {        this.$confirm("确认关闭？")          .then(\_ => {            done();          })          .catch(\_ => {});      }    }  };  </script> |

### 修改router/index.js

### 效果

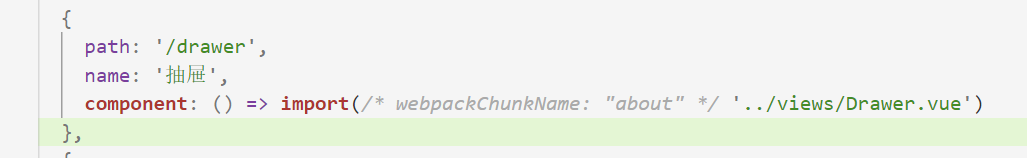


## 抽屉

### 创建view/Drawer.vue

|  |
| --- |
| <template>    <div>      <el-radio-group *v-model*="direction">        <el-radio *label*="ltr">从左往右开</el-radio>        <el-radio *label*="rtl">从右往左开</el-radio>        <el-radio *label*="ttb">从上往下开</el-radio>        <el-radio *label*="btt">从下往上开</el-radio>      </el-radio-group>      <el-button @*click*="open" *type*="primary" *style*="margin-left: 16px;">点我打开</el-button>      <el-drawer        :*visible*.*sync*="drawer"        :*direction*="direction"      >        <span>我来啦!</span>      </el-drawer>    </div>  </template>  <script>  export default {  **data**() {      return {        direction: "rtl",  *//默认不打开抽屉*        drawer: false,      };    },    methods: {  **open**() {        this.drawer = true;      },    },  };  </script> |

### 修改router/index.js



### 效果



## 表格

### 创建view/Table.vue

|  |
| --- |
| <template>    <el-table :data="tableData" border style="width: 100%">      <el-table-column fixed prop="date" label="日期"></el-table-column>      <el-table-column prop="name" label="姓名" ></el-table-column>      <el-table-column prop="province" label="省份" ></el-table-column>      <el-table-column prop="city" label="市区" ></el-table-column>      <el-table-column prop="address" label="地址" ></el-table-column>      <el-table-column prop="zip" label="邮编" ></el-table-column>      <el-table-column fixed="right" label="操作" >        <template slot-scope="scope">          <el-button @click="handleClick(scope.row)" type="text" size="small">查看</el-button>          <el-button type="text" size="small">编辑</el-button>        </template>      </el-table-column>    </el-table>  </template>  <script>  export default {    name: 'Table',    methods: {      handleClick(row) {        console.log(row);      }    },    data() {      return {        tableData: [          {            date: "2016-05-02",            name: "王小虎",            province: "上海",            city: "普陀区",            address: "上海市普陀区金沙江路 1518 弄",            zip: 200333          },          {            date: "2016-05-04",            name: "王小虎",            province: "上海",            city: "普陀区",            address: "上海市普陀区金沙江路 1517 弄",            zip: 200333          },          {            date: "2016-05-01",            name: "王小虎",            province: "上海",            city: "普陀区",            address: "上海市普陀区金沙江路 1519 弄",            zip: 200333          },          {            date: "2016-05-03",            name: "王小虎",            province: "上海",            city: "普陀区",            address: "上海市普陀区金沙江路 1516 弄",            zip: 200333          }        ]      };    }  };  </script> |

### 修改router/index.js

### 效果



# 【掌握】axios 的使用

axios是一个基于 promise 的 HTTP 库，在vue中axios是比较常用的网络请求方法。

### 安装

npm install axios -S

## 在main.js配置全局

|  |
| --- |
| import axios from 'axios'  Vue.prototype.$axios = axios |

## 使用在页面

|  |
| --- |
| this.$axios({    url: "",    method: "get",    params: {}  }).then(res => {    console.log(res)  }) |

也可以先进行封装后调用

## 封装

先新建一个network文件夹再建一个request.js

|  |
| --- |
| import axios from 'axios'  export function request(config) {      return new Promise((resolve, reject) => {          创建axios实例          const instance = axios.create({              baseURL: '',          })          // 发送网络请求          instance(config).then(res => {              resolve(res)          }).catch(err => {              reject(err)          })      })  } |

## 简写

|  |
| --- |
| import axios from 'axios'  export function request(config) {      // 创建axios实例      const instance = axios.create({          baseURL: '',      })      // 发送网络请求      return instance(config)  } |

## 使用

|  |
| --- |
| import {request} from '../network/request'  request({        url:'',        params: {}      }).then(res => {        console.log(res)  }) |

# 【掌握】vue+axios+eleUI完成单表的CRUD

## 数据库

## 后端接口编写

## 使用Vue-cli创建项目

## 安装elementUI

## 编写页面

## 项目结构优化

# 【掌握】vue-element-admin的使用