**浙江民泰智慧社区平台开发文档**

[1、项目搭建 2](#_Toc55812105)

[1.1、VPN环境配置 2](#_Toc55812106)

[1.2、GitLab账号注册 2](#_Toc55812107)

[1.3、node.js本地环境搭建 3](#_Toc55812108)

[1.4、开发工具Visual Studio Code 3](#_Toc55812109)

[1.5、GitLab项目查看 3](#_Toc55812110)

[1.6、项目分支查看 4](#_Toc55812111)

[1.7、项目克隆 4](#_Toc55812112)

[1.8、项目搭建 5](#_Toc55812113)

[2、项目结构 7](#_Toc55812114)

[2.1、根目录下vue.config.js文件 7](#_Toc55812115)

[2.2、根目录下的src文件 8](#_Toc55812116)

[2.2.1、main.js文件、App.vue文件 8](#_Toc55812117)

[2.2.2、api文件 8](#_Toc55812118)

[2.2.3、assets文件 9](#_Toc55812119)

[2.2.4、router文件 9](#_Toc55812120)

[2.2.5、store文件 9](#_Toc55812121)

[2.2.6、views文件 9](#_Toc55812122)

[3、页面开发 10](#_Toc55812123)

[3.1、新建Vue文件 10](#_Toc55812124)

[3.2、文件结构 11](#_Toc55812125)

[3.2.1、三大结构 11](#_Toc55812126)

[3.2.2、路由配置 11](#_Toc55812127)

[3.2.3、页面的访问 11](#_Toc55812128)

[3.2.4、Vue页面的生命周期函数 11](#_Toc55812129)

[3.2.5、计算属性computed的使用 12](#_Toc55812130)

[3.2.6、watch事件 13](#_Toc55812131)

[3.2.7、自定义组件 13](#_Toc55812132)

[3.2.8、父组件向子组件传递数据 14](#_Toc55812133)

[3.2.9、在子组件中触发父组件的方法并传递参数 14](#_Toc55812134)

[3.2.10、ref属性的使用 15](#_Toc55812135)

[3.2.11、组件中slot插槽的使用 16](#_Toc55812136)

[3.3、开发过程中的常用路由跳转方式 17](#_Toc55812137)

[3.3.1、最常用的push跳转方式 17](#_Toc55812138)

[3.3.2、replace跳转方式 18](#_Toc55812139)

[3.3.3、go跳转方式 18](#_Toc55812140)

[3.4、项目开发中的数组操作方法 18](#_Toc55812141)

[3.4.1、join() 18](#_Toc55812142)

[3.4.2、reverse() 18](#_Toc55812143)

[3.4.3、concat() 18](#_Toc55812144)

[3.4.5、slice() 18](#_Toc55812145)

[3.4.6、push() 18](#_Toc55812146)

[3.4.7、pop() 19](#_Toc55812147)

[3.4.8、shift() 19](#_Toc55812148)

[3.4.9、unshift() 19](#_Toc55812149)

[3.4.10、toString()和toLocaleString() 19](#_Toc55812150)

[3.5、数组中的高阶函数 19](#_Toc55812151)

[3.5.1、sort() 19](#_Toc55812152)

[3.5.2、forEach() 19](#_Toc55812153)

[3.5.3、map() 19](#_Toc55812154)

[3.5.4、filter() 19](#_Toc55812155)

[3.5.5、every()和some() 20](#_Toc55812156)

[3.5.6、 reduce()和reduceRight() 20](#_Toc55812157)

[3.6、项目开发中常用的ES6语法 20](#_Toc55812158)

[3.6.1、形参的默认赋值 20](#_Toc55812159)

[3.6.2、形参与解构赋值结合 20](#_Toc55812160)

[3.6.3、可变参数 21](#_Toc55812161)

[3.6.4、扩展运算符 21](#_Toc55812162)

[3.6.5、promise语法 21](#_Toc55812163)

[3.6.6、export暴露语法 25](#_Toc55812164)

[3.6.7、import倒入语法 26](#_Toc55812165)

[3.6.8、项目常用的async函数 26](#_Toc55812166)

[3.6.9、项目中常用的await表达书 28](#_Toc55812167)

[3.6.10、async和await获得ajax请求获得数据 28](#_Toc55812168)

[4、项目使用的使用的组件框架--View UI 29](#_Toc55812169)

[4.1、组件框架的使用(以分页组件page为例) 29](#_Toc55812170)

[4.2、组件参数api 29](#_Toc55812171)

[4.3、组件触发的事件 29](#_Toc55812172)

[4.4、组件的插槽使用 29](#_Toc55812173)

# 1、项目搭建

## 1.1、VPN环境配置

详情请见如下附件



**注意：**项目开发测试过程中，VPN始终保持连接状态

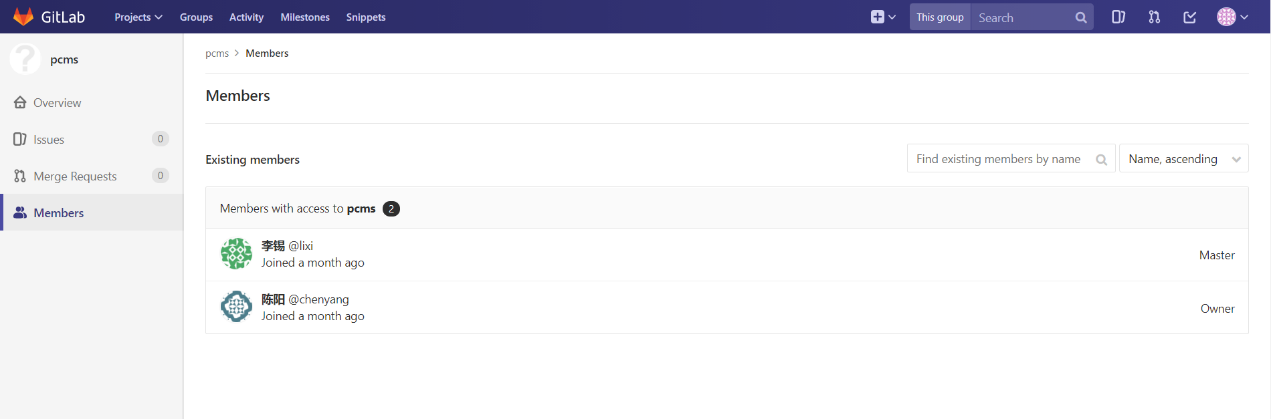
## 1.2、GitLab账号注册

申请GitLab账号，并要求项目管理人员将本人拉入到GitLab的项目团队中，开发人员方可拉取项目进行开发

GitLab网址：

<http://192.168.15.125/users/sign_in>

## 1.3、node.js本地环境搭建

node.js版本安装官网

<http://nodejs.cn/download/current/>

node.js环境搭建详细步骤如下

<https://segmentfault.com/a/1190000023390756>

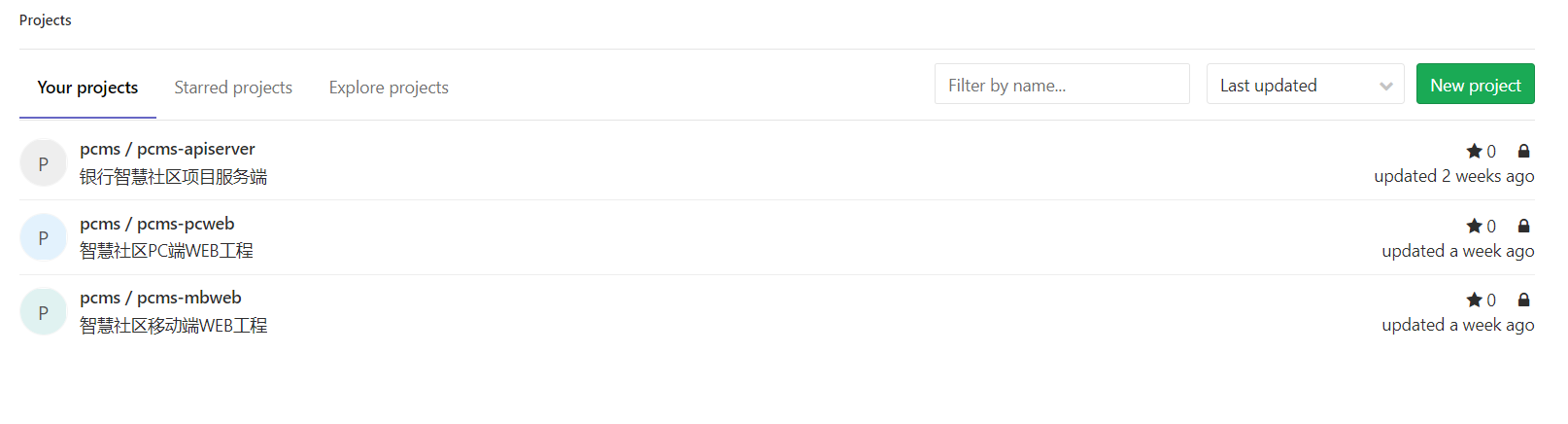
## 1.4、开发工具Visual Studio Code

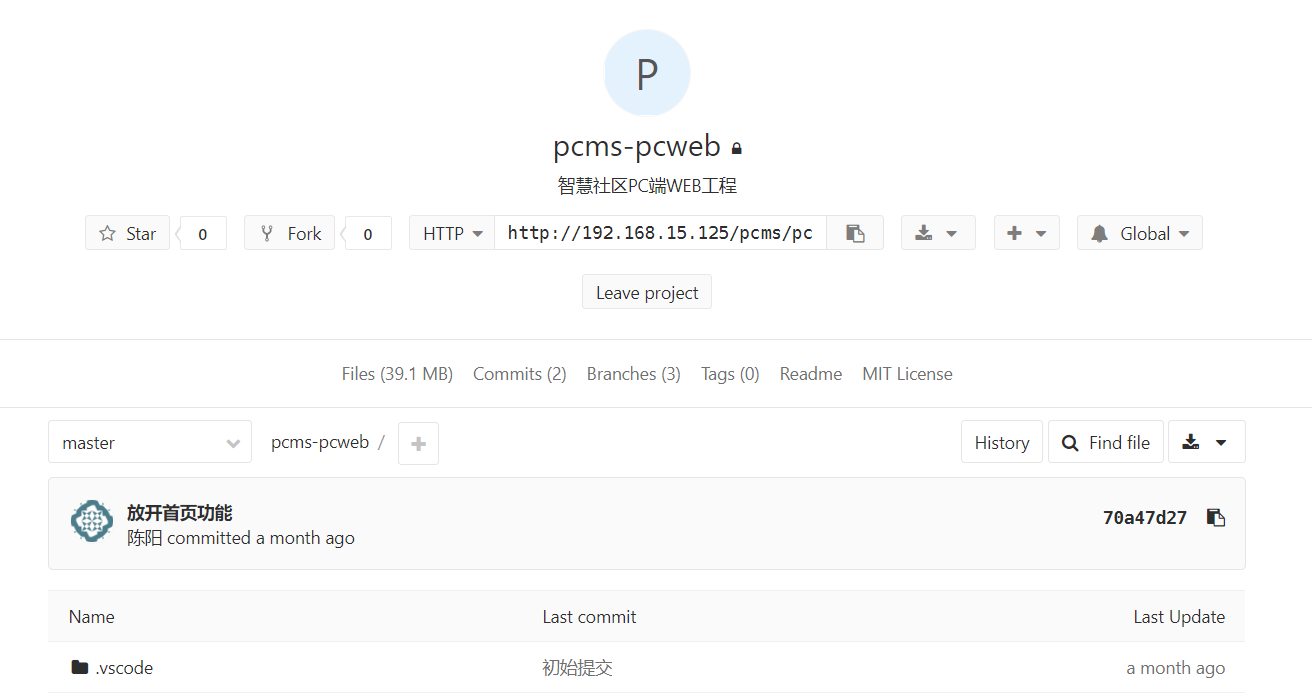
下载及安装地址：

<https://blog.csdn.net/qq_42367703/article/details/88074154>

## 1.5、GitLab项目查看

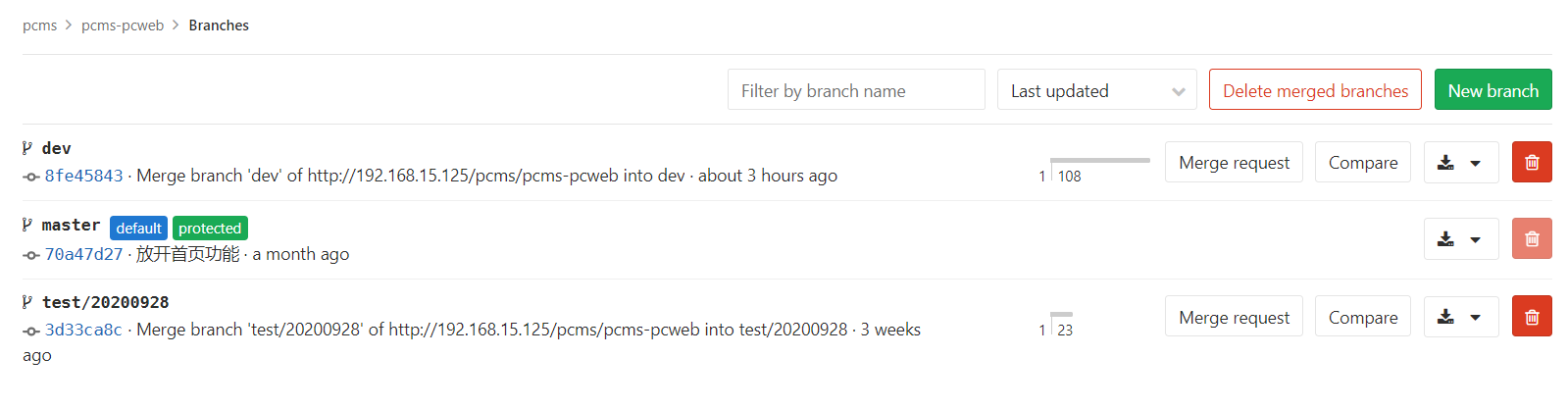
注意：本文档所介绍的项目----智慧社区PC端WEB工程





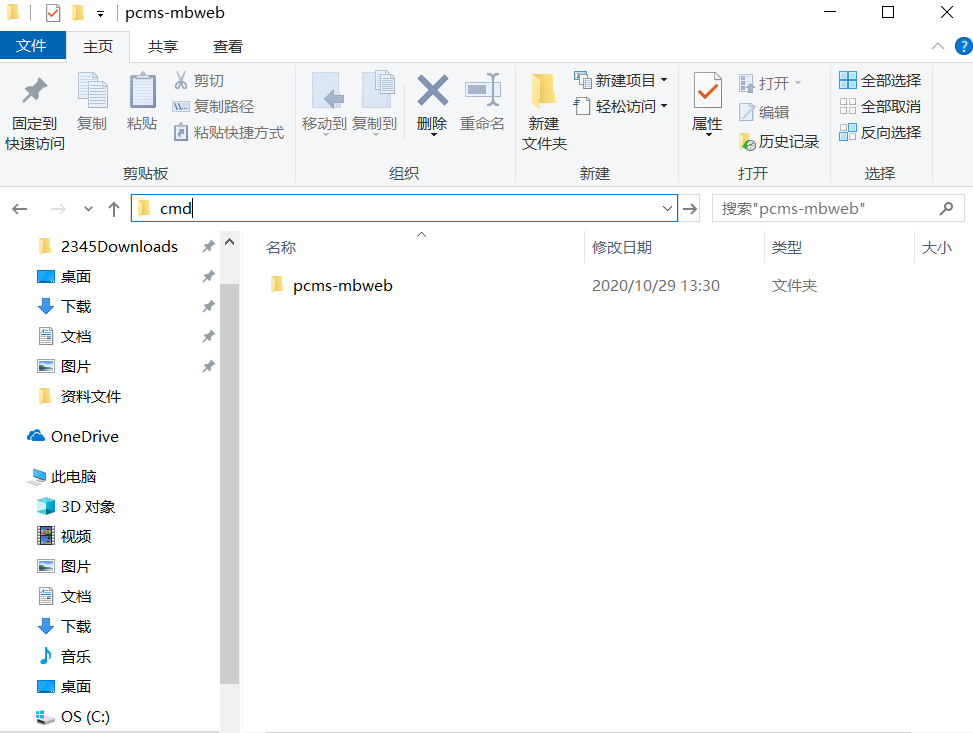
## 1.6、项目分支查看

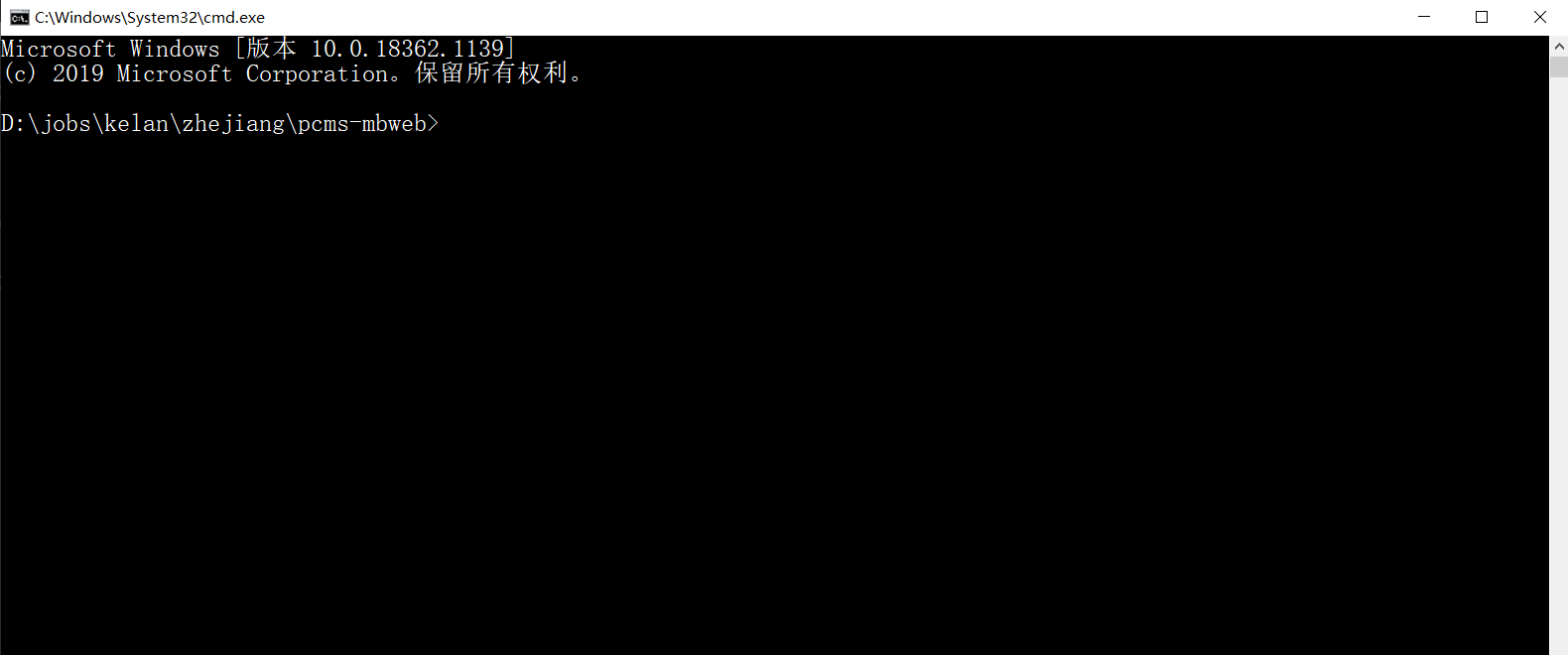
注意：本次项目开发运维在**dev**分支上进行



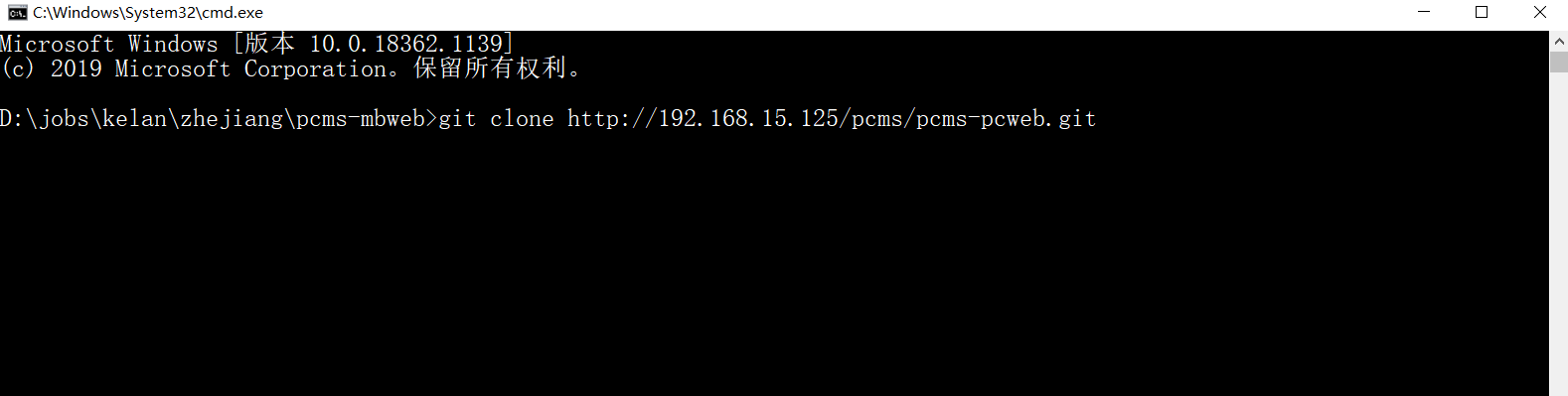
## 1.7、项目克隆

①打开一个文件夹，在文件夹的路径输入框里面输入cmd,然后回车





②在里面输入git clone <http://192.168.15.125/pcms/pcms-pcweb.git>然后回车



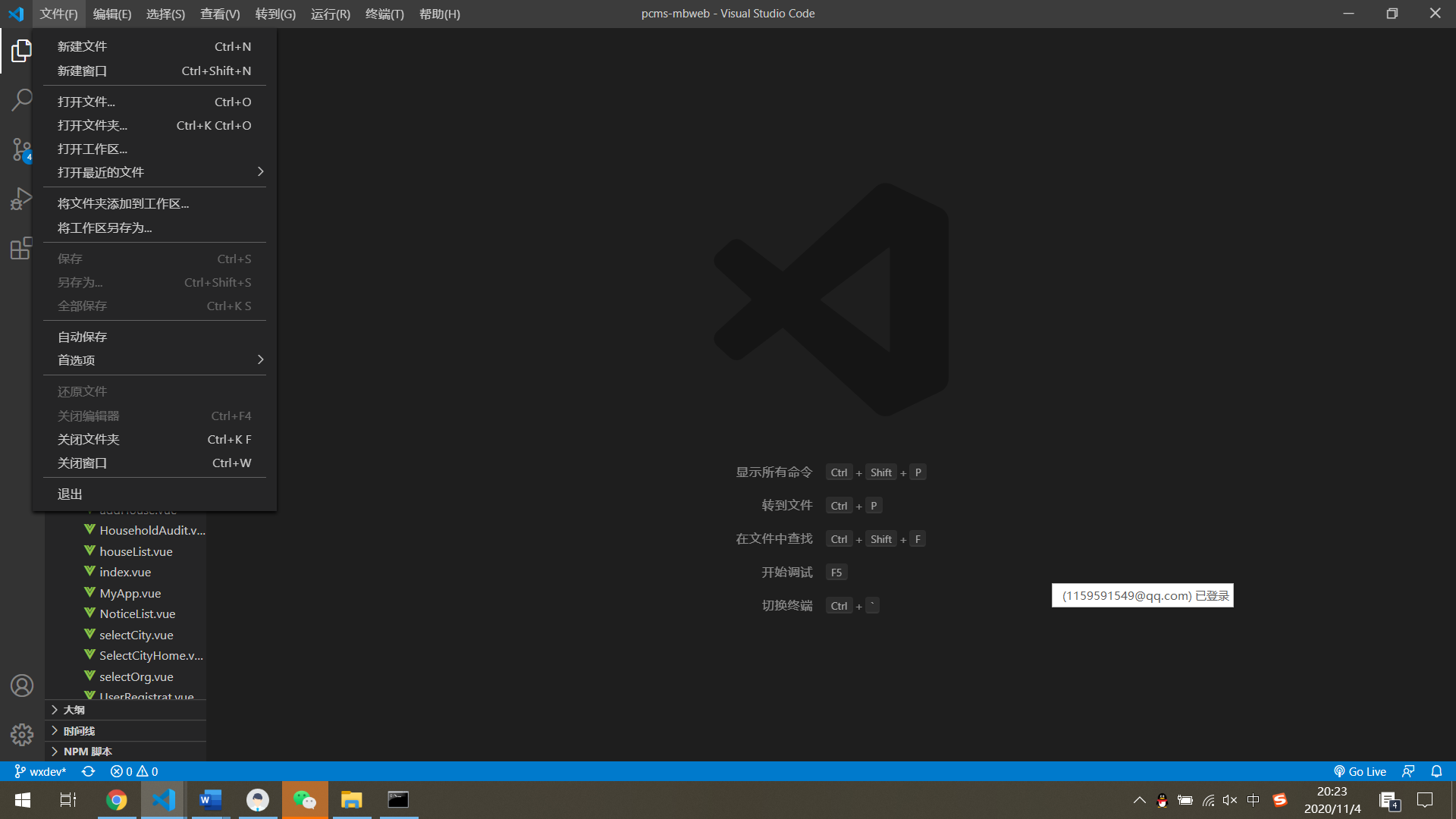
等待执行完毕且没有报错，项目才克隆成功

## 1.8、项目搭建

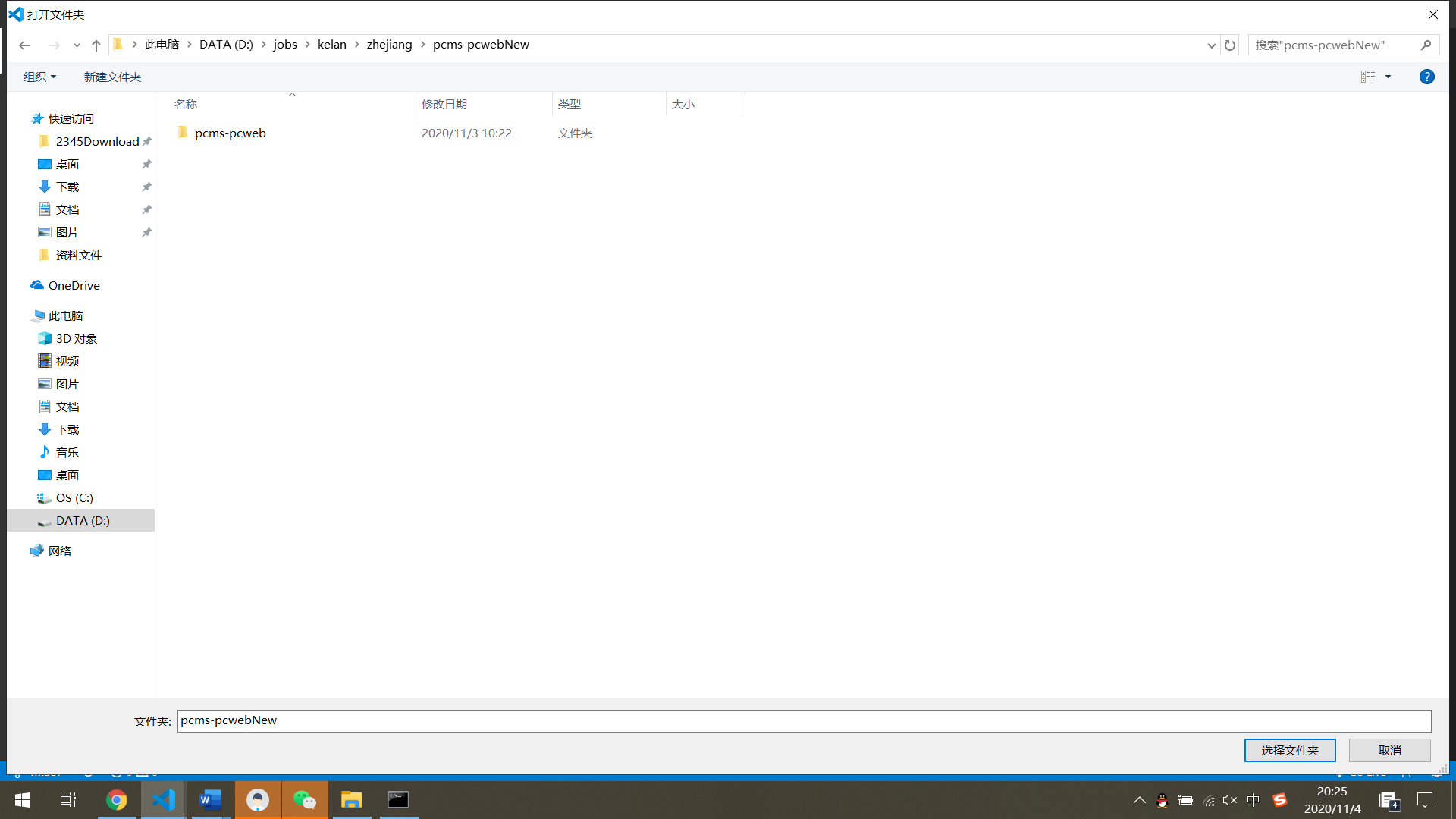
①打开Vscode



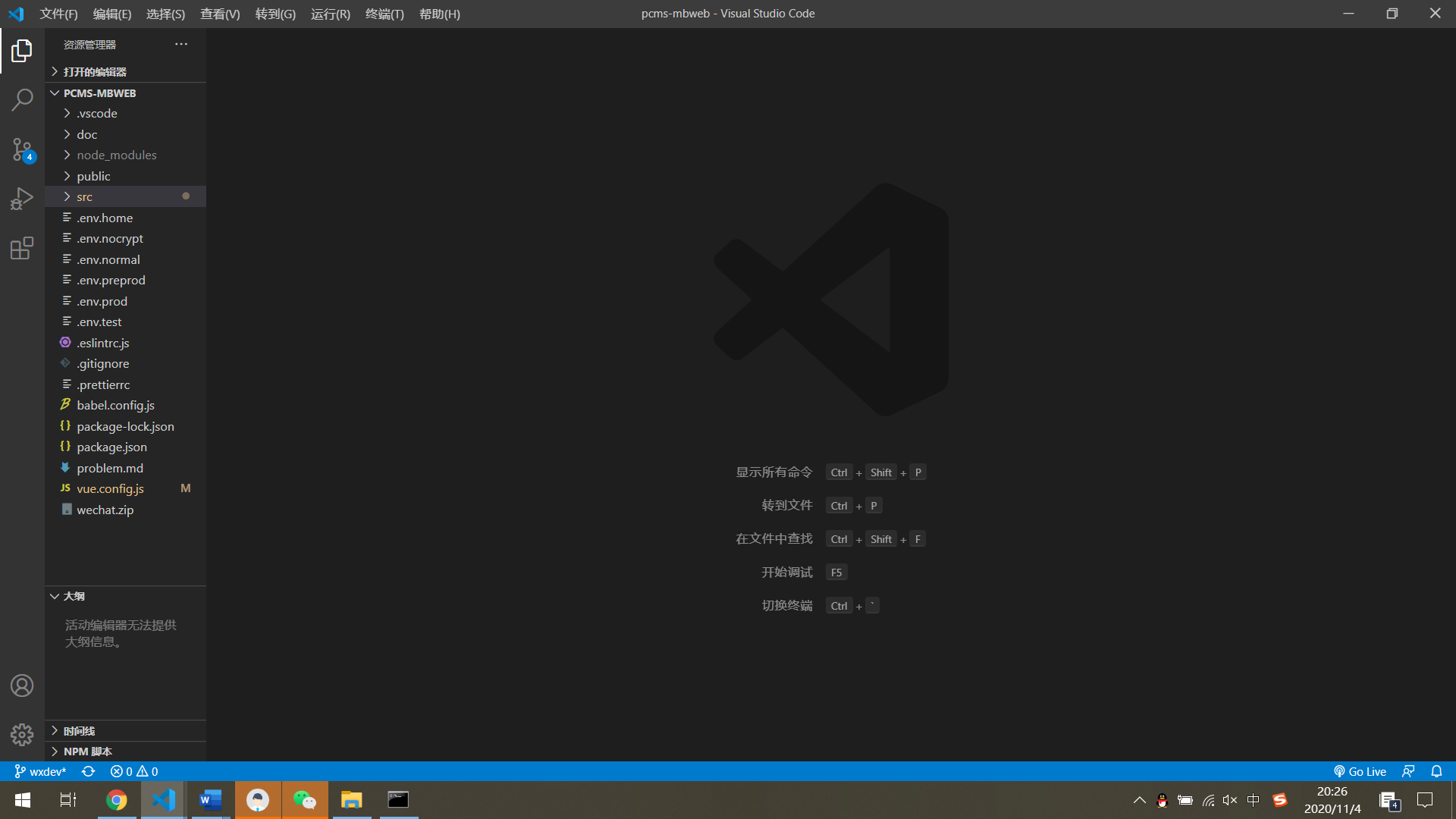
②点击左上角文件，然后点击打开文件夹



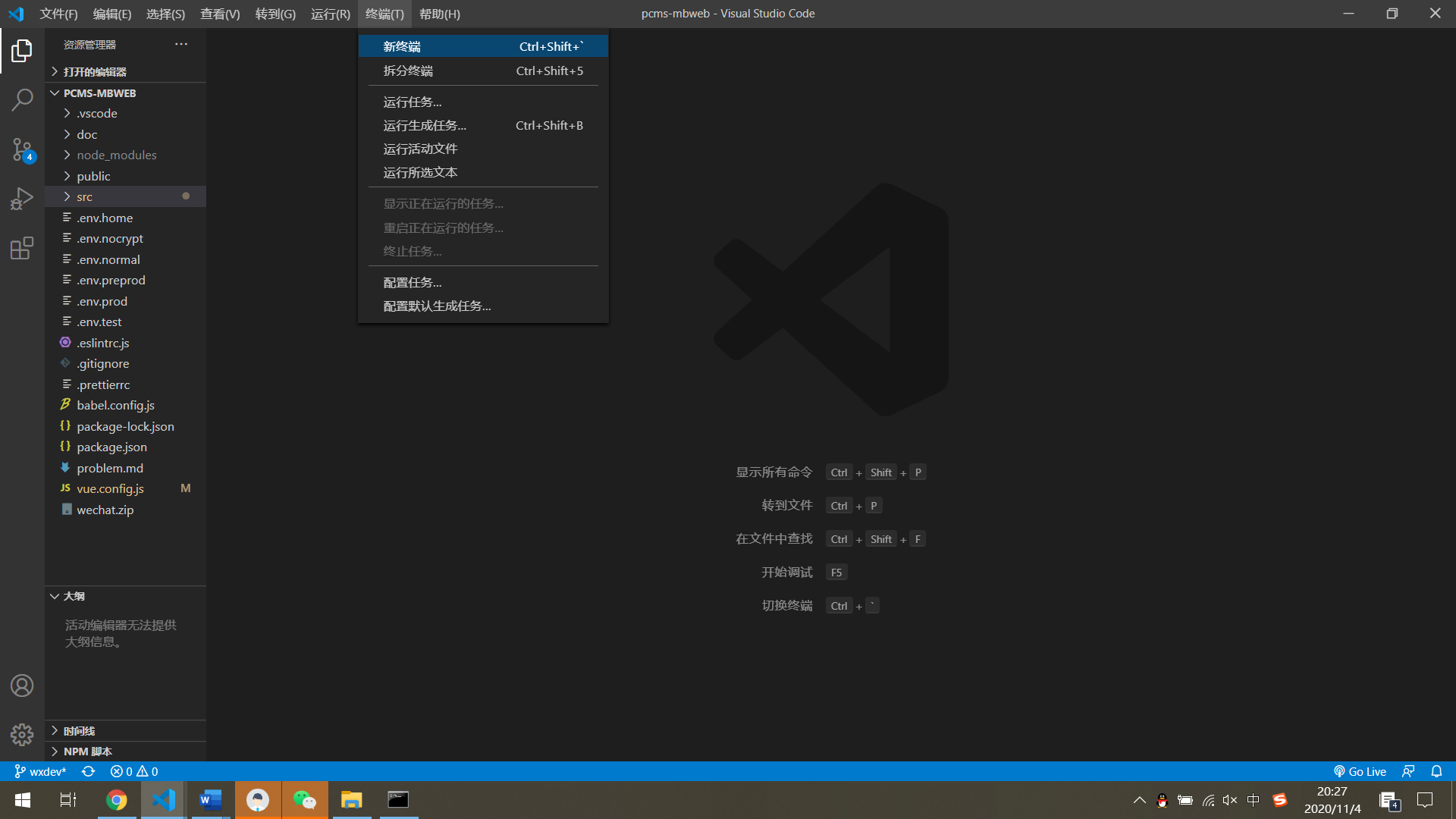
③找到刚才下载的文件选择即可



④项目导入到状态

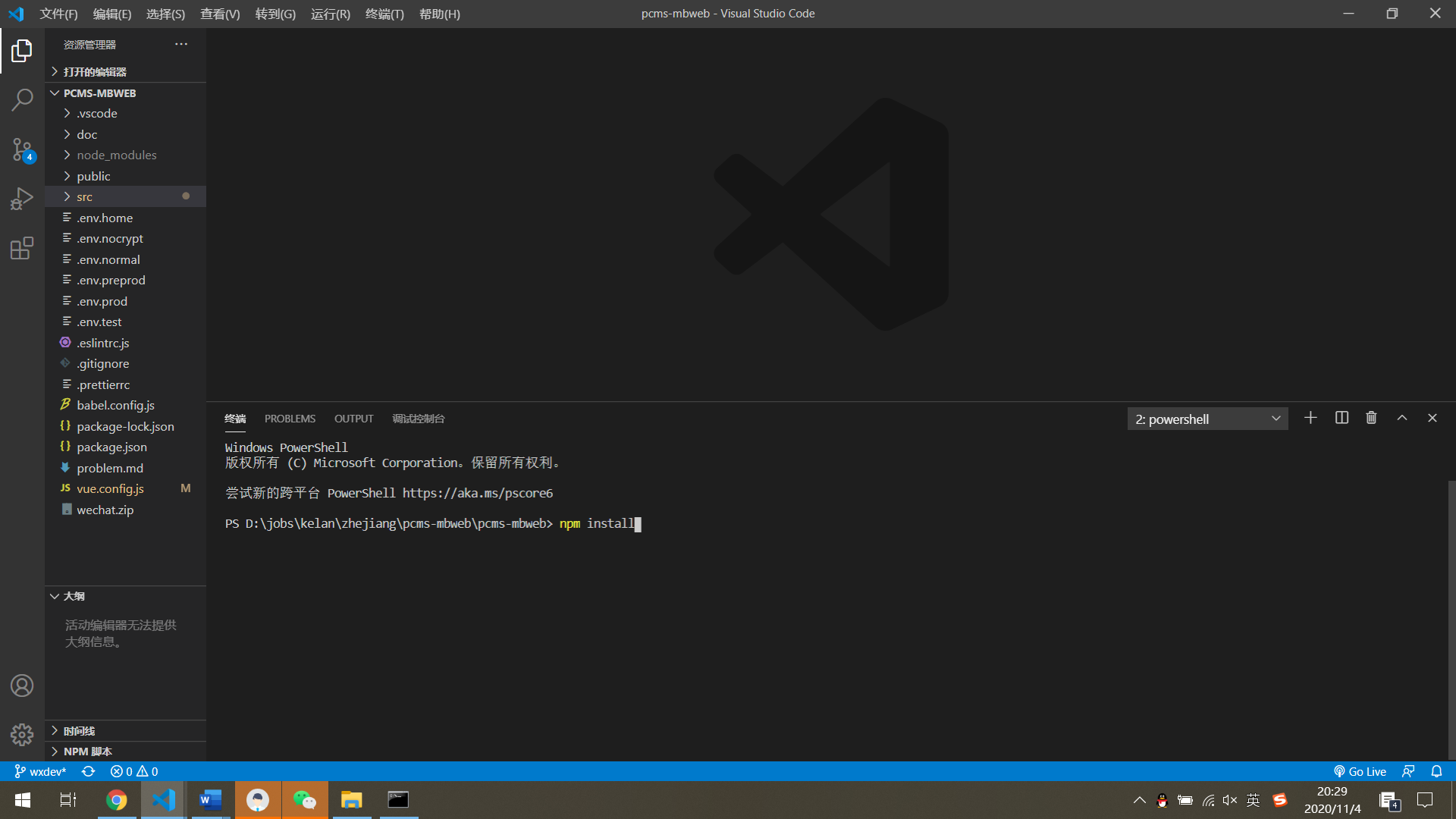


⑤打开一个新的终端

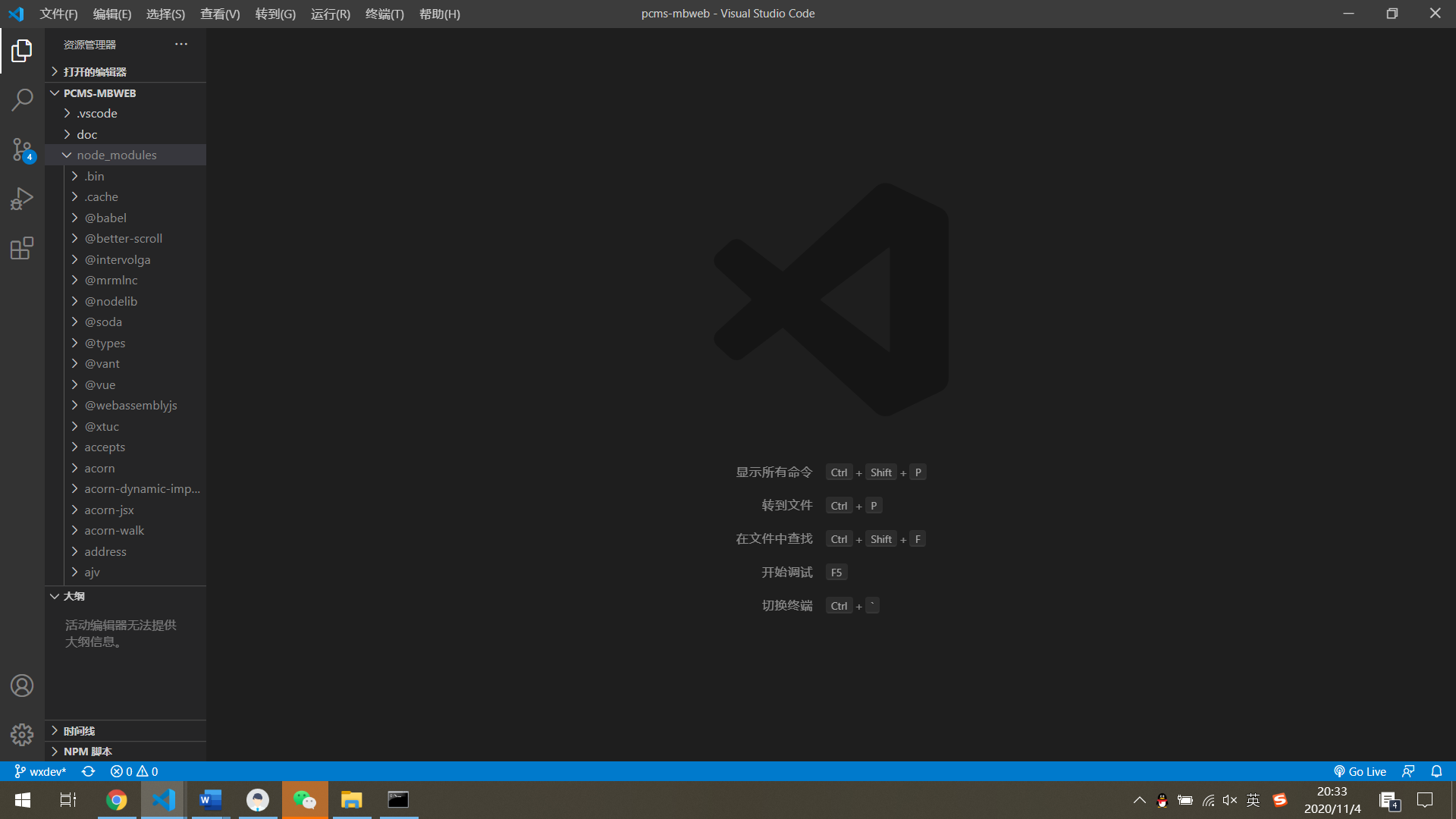


⑥在里面执行npm install命令

注意：本步骤时间较长



执行完毕，会在项目中生成node\_modules文件夹，该文件下存放此项目所使用的依赖



⑦在终端中查看所在分支，如果不在dev分支请切换分支到dev上进行开发

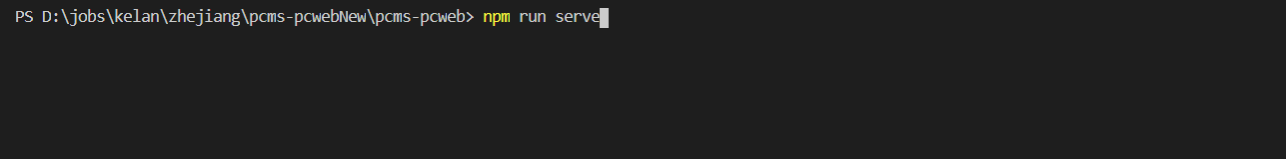


⑧分支切换指令

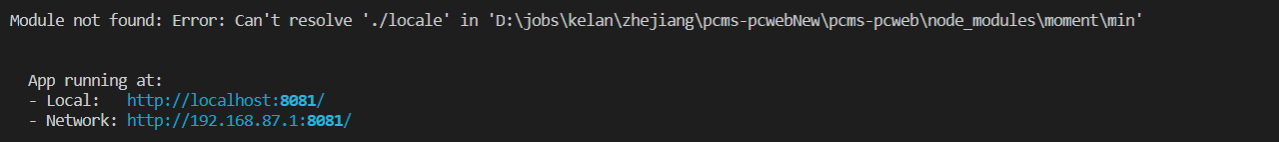


⑨项目启动

在终端中输入npm run serve



等到出现下图时，项目启动成功，页面会成功跳转到登陆页面



登陆界面



登陆账号密码为

用户名：18888888888

密码：Zy@123456

# 2、项目结构

## 2.1、根目录下vue.config.js文件

配置项目信息，最主要的是配置项目所依赖的服务环境

配置位置：代码74-84行，修改IP地址即可连接不同的服务



## 2.2、根目录下的src文件

用于存放开发过程中的代码和资源



### 2.2.1、main.js文件、App.vue文件

main.js文件用来引入项目所需资源

APP.vue文件为程序的入口

### 2.2.2、api文件

①作用：引入项目所使用的接口

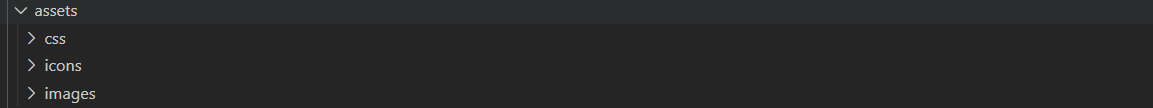
②文件结构



③.js文件的创建

|  |
| --- |
| 1、在文件的开头引入axios  import axios from '@/libs/api.request'  2、将请求的接口封装成参数并暴露出去  export function functionName(data) {    let url = `接口的地址`    return axios.request({      url,      data    })  }  **例子**    **注意：**  export关键字将这个方法暴露出去  url后端接口的地址  date请求接口需要的参数  functionName为函数名  3、接口函数的使用  在需要使用的文件中引入接口：  import { 接口函数名 } from '接口函数路径'  **例子**    在使用的接口函数的时候以调用函数的方式使用即可 |

### 2.2.3、assets文件



CSS中存放公共的样式

icons中放一些图标

images放静态的图片资源

### 2.2.4、router文件

配置存放页面的路由

### 2.2.5、store文件

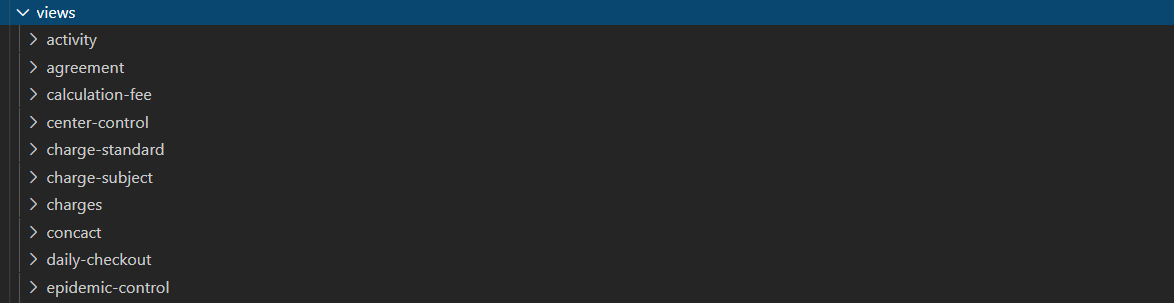
主要使用Vuex进行状态管理

Vuex的使用方法详见文件夹中的Vuex状态管理文档



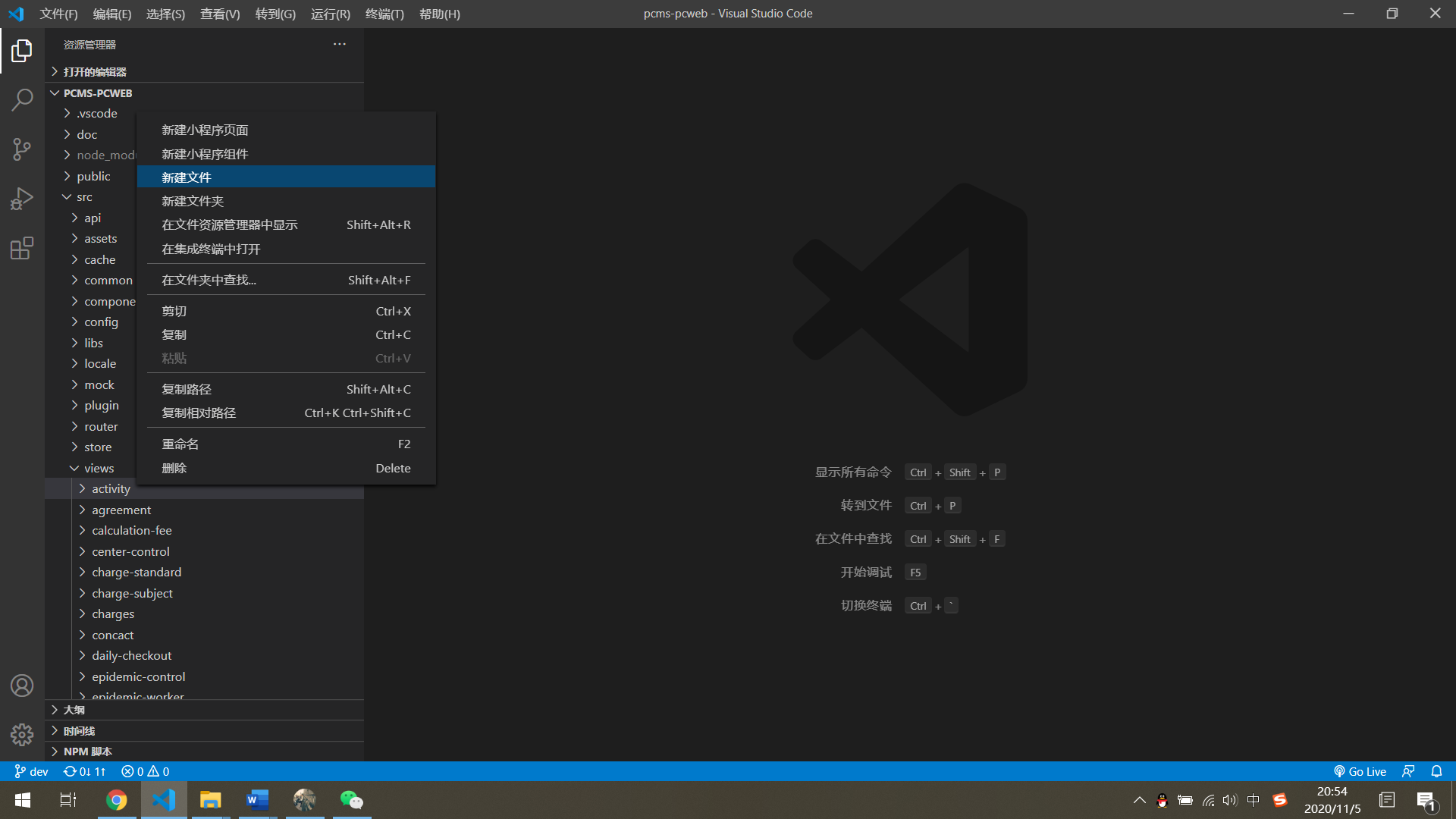
### 2.2.6、views文件

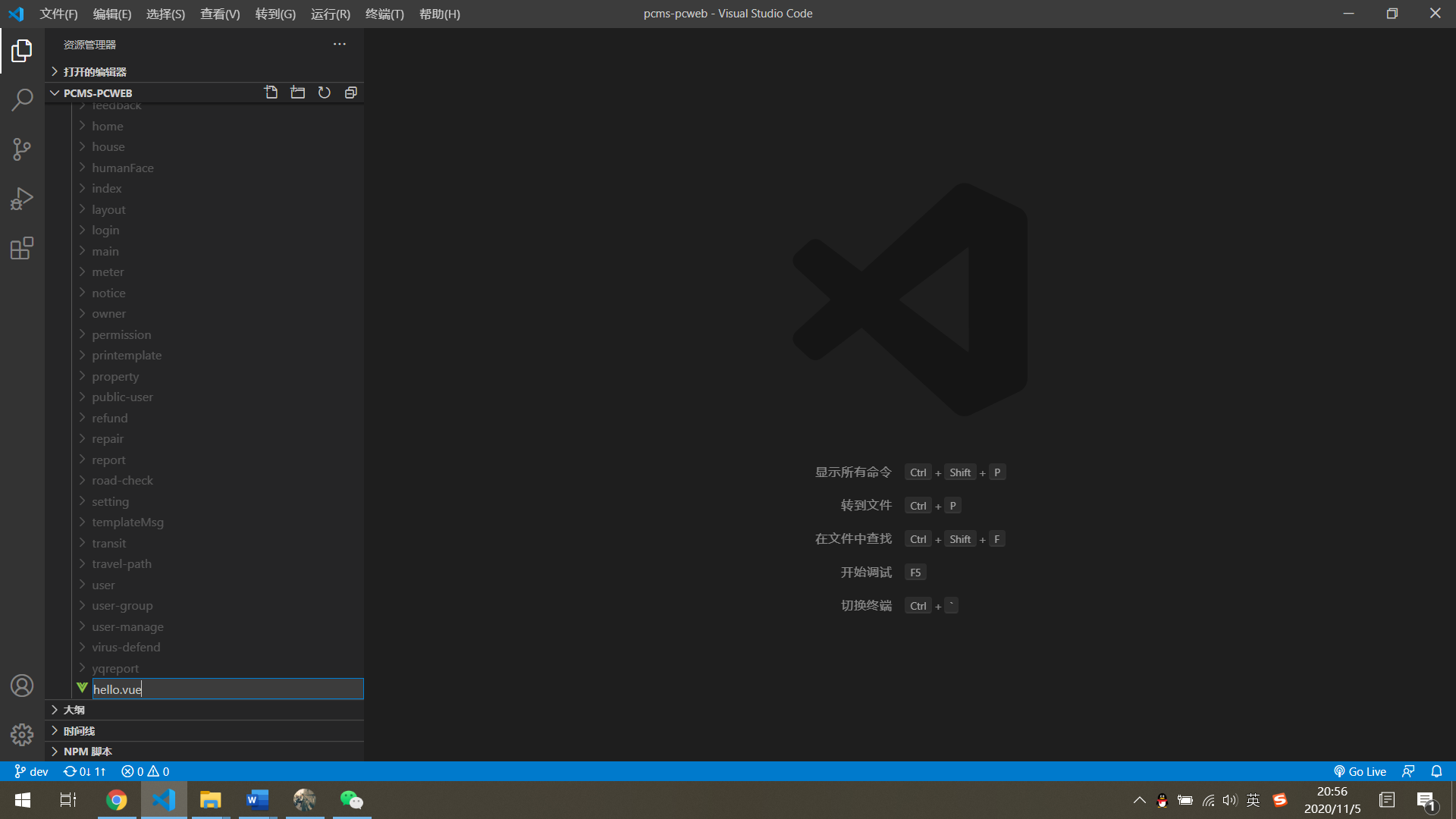
存放页面代码



# 3、页面开发

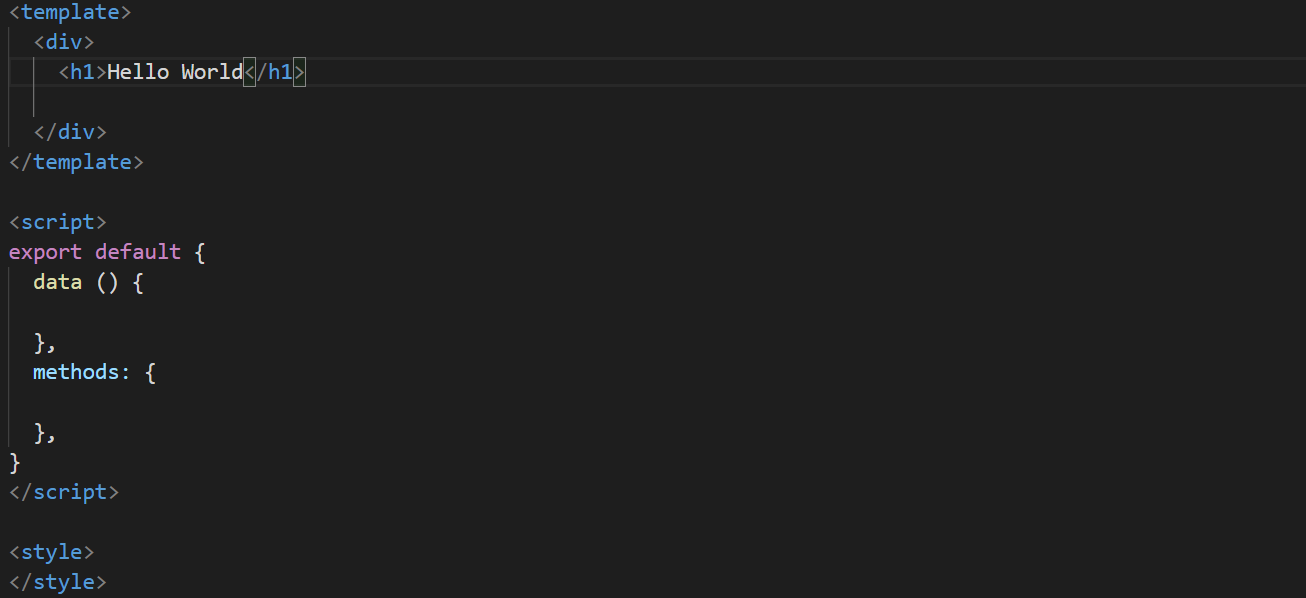
## 3.1、新建Vue文件





## 3.2、文件结构

### 3.2.1、三大结构



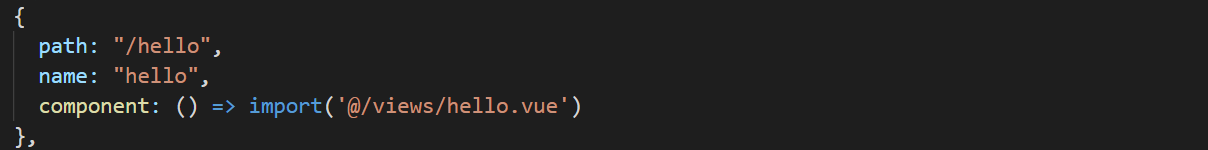
template标签中编写HTML代码

**注意：**在template标签中只能有一个根目录

script标签中编写JavaScript代码

style标签中编写CSS代码

### 3.2.2、路由配置



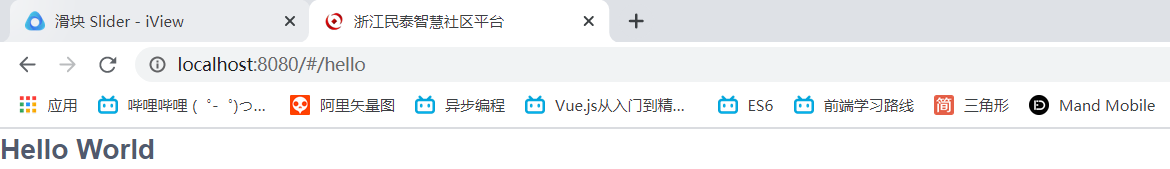
**path：**表示访问的路径

**name：**表示访问的名称

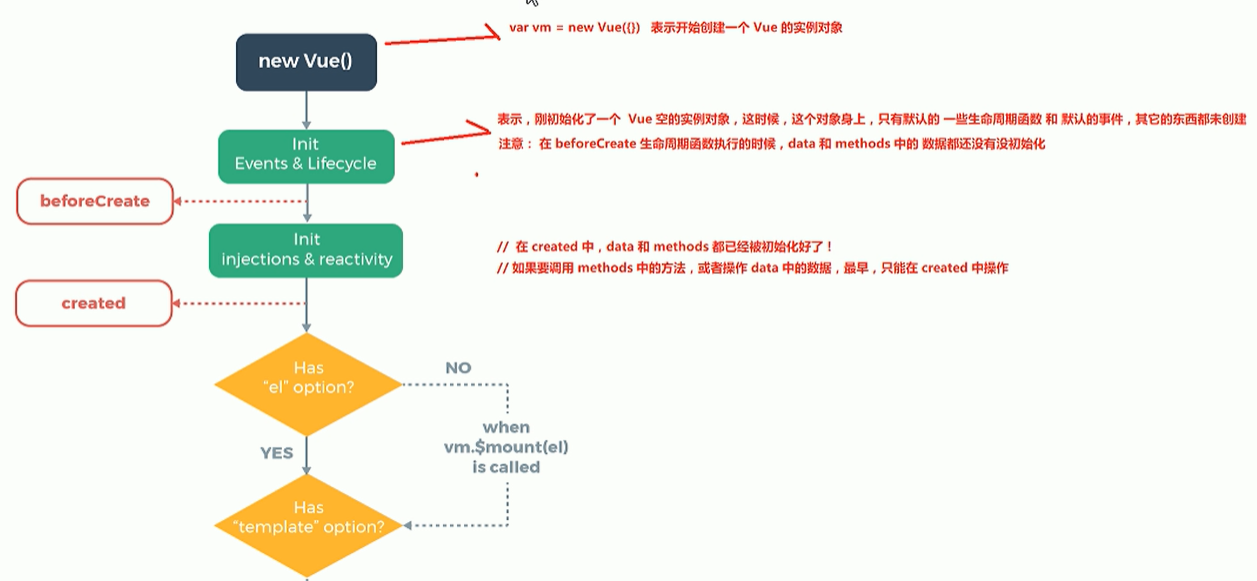
**component：**表示创建的页面路径

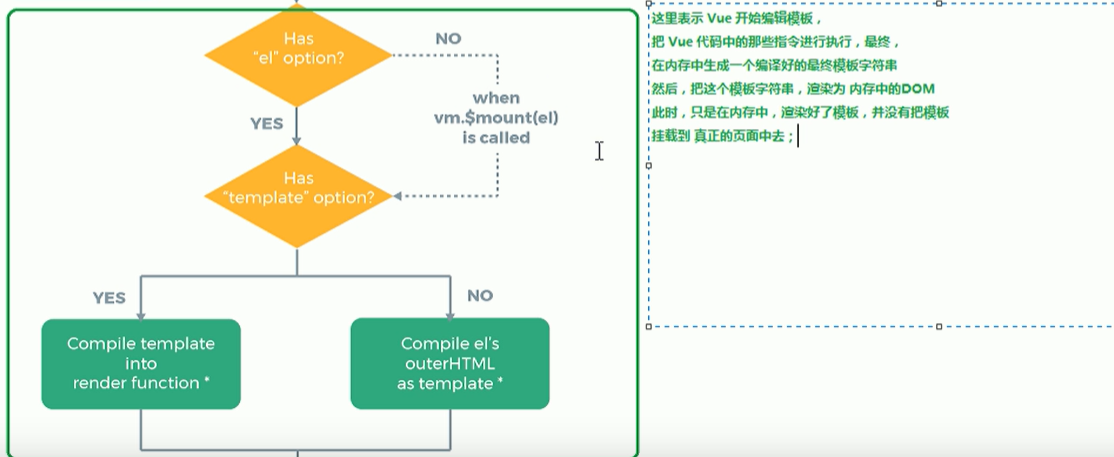
**@：**符号表示从src目录开始

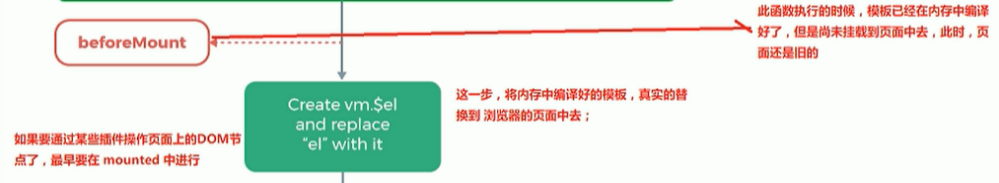
### 3.2.3、页面的访问

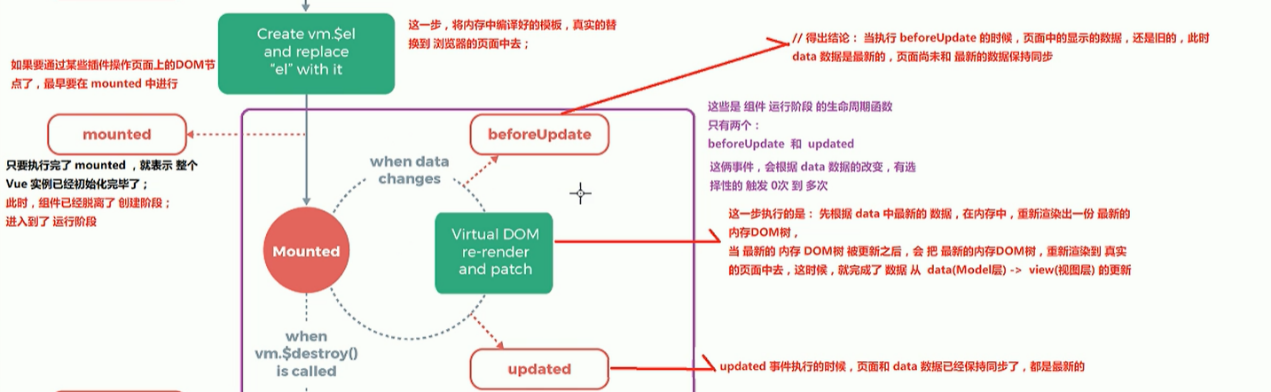


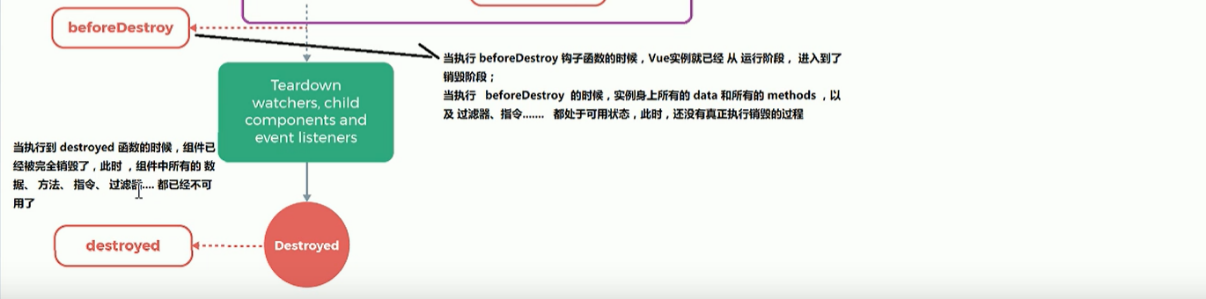
### 3.2.4、Vue页面的生命周期函数











注意：created方法和mounted方法在在页面加载的时候会执行，一般对于页面需要使用的数据都会放在这两个方法中执行

### 3.2.5、计算属性computed的使用

computed:在 computed 中，可以定义一些属性，这些属性，叫做**计算属性**，计算属性的本质，就是一个方法，只不过，我们在使用这些计算属性的时候，是把它们的名称，直接当作属性来使用的，并不会把计算属性，当作方法去调用。

**注意1**：计算属性，在引用的时候，一定不要加()去调用，直接把它当作 普通属性去使用就好了。

**注意2：**只要计算属性，这个 function内部，所用到的任何 data 中的数据发送了变化，就会立即重新计算这个计算属性的值。

**注意3：**计算属性的求值结果，会被缓存起来，方便下次直接使用;如果‘计算属性方法中，所以来的任何数据，都没有发生过变化，则不会重新对计算属性求值。

|  |
| --- |
| **在computed中定义计算属性**  computed:{  functionName(){  //该函数必须要有返回值data  return data  }  }  **计算属性的使用**  直接在需要调用的地方调用计算属性的函数名即可  <input v-model=‘functionName’></input> |

### 3.2.6、watch事件

监测某一个值的变化，当这个值变化时，执行watch中的方法

watch:{

text1:function(){},//监测text1的变化，当text1变化时，执行后面的函数

text2:function(){}//监测text2的变化，当text2变化时，执行后面的函数

}

### 3.2.7、自定义组件

|  |
| --- |
| 1、新建.vue文件，文件结构和普通的vue文件结构一致      2、在需要使用组件的页面导入组件  import hello from './components/hello'  3、在当前页面的components属性中注册组件  components对象所在的位置和当前页面data对象处在同一目录    components: {  hello  }  4、组件的使用  <hello></hello>  5、页面显示 |

### 3.2.8、父组件向子组件传递数据

|  |
| --- |
| 1、在父组件中绑定需要传给子组件的数据。绑定形式如下：    2、在子组件中的props属性中定义父组件传过来的数据  **注意：**在定义变量的时候在type中指明变量的类型，在default中指明变量的默认值，当变量的类型是数组时，默认值要是函数的形式，具体如下：    3、变量的使用  在子组件中使用父组件传过来的数据像使用data中定义的数据一样使用    4、页面的展示 |

### 3.2.9、在子组件中触发父组件的方法并传递参数

|  |
| --- |
| 1、子组件页面按钮    2、子组件页面代码    3、点击按钮触发子组件中btnClick事件  注意：通过this.$emit(“父组件方法名”,参数)    4、父组件的方法  注意：t通过触发父组件中的fun方法，执行fatherFun方法      5、页面显示 |

### 3.2.10、ref属性的使用

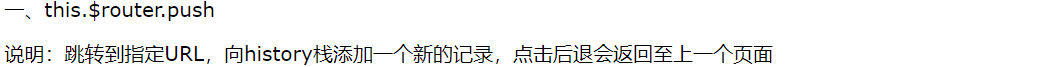
|  |
| --- |
| 作用：父页面中，通过ref属性调用子组件中的方法和数据    访问子组件中的数据和方法    子组件的数据和方法      页面展示 |

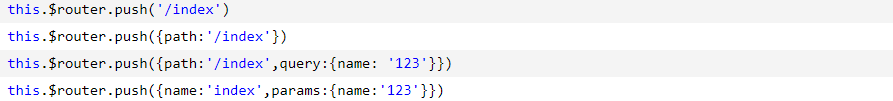
### 3.2.11、组件中slot插槽的使用

|  |
| --- |
| **注意：**本例所使用的组件是上面已经定义好的<hello></hello>组件  **作用：**在父页面中调用子组件时，在子组件的标签中写HTML代码是不显示的，slot插槽就是解决这个问题  **演示：**  父页面    子组件中没有slot插槽    页面显示    子组件中有slot插槽    页面显示    1、匿名插槽  匿名插槽的演示已经在上面展示  2、具名插槽  注意插槽的名字和排列的顺序  父组件中的插槽的使用    子组件中插槽的使用    页面展示    3、作用域插槽  **介绍：**作用于插槽主要将样式和数据分开，样式由父组件决定，数据由子组件决定  子组件的HTML代码    子组件数据    父组件的调用    页面显示 |

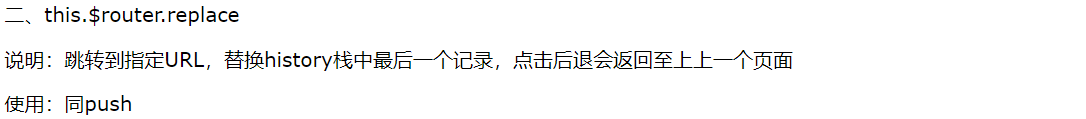
## 3.3、开发过程中的常用路由跳转方式

### 3.3.1、最常用的push跳转方式

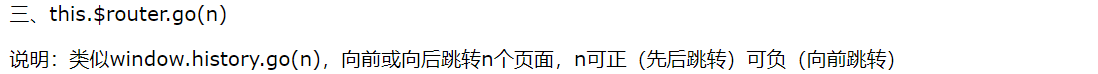


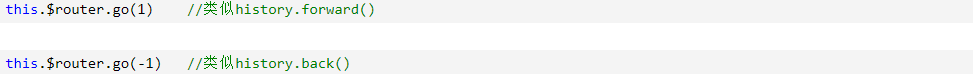


### 3.3.2、replace跳转方式



### 3.3.3、go跳转方式

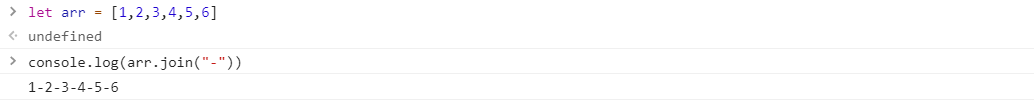




## 3.4、项目开发中的数组操作方法

### 3.4.1、join()

功能：将数组中所有元素都转化为字符串并连接在一起。

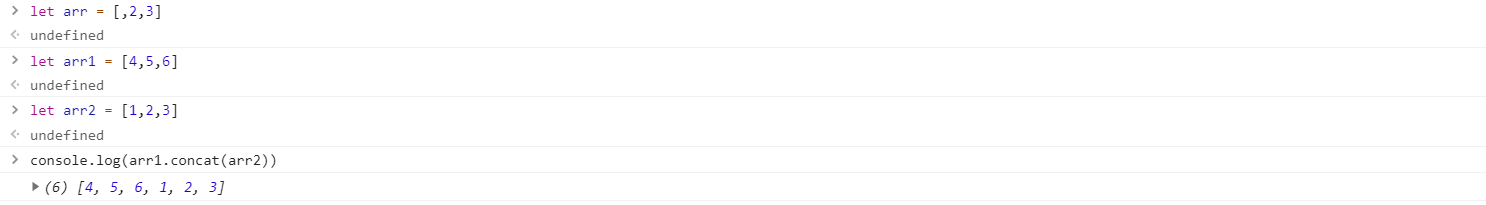


### 3.4.2、reverse()

功能：将数组中的元素颠倒顺序。

### 3.4.3、concat()

功能：数组拼接的功能 ,返回新数组，原数组不受影响。



### 3.4.5、slice()

截取数组生成新数组，原数组不受影响。

返回的数组包含第一个参数指定的位置和所有到但不含第二个参数指定位置之间的所有元素。

如果为负数，表示相对于数组中最后一个元素的位置。如果只有一个参数，表示到数组末尾。

|  |
| --- |
| var aa = [1,2,3,4,5,6];  console.log(aa.slice(2)); //[3,4,5,6]  console.log(aa.slice(2,8)); //[3,4,5,6] 超过最大长度，只显示到最后结果  console.log(aa.slice(2,5)); //[3,4,5]  console.log(aa.slice(2,-1)); //[3,4,5] 相对于倒数第一个之前  console.log(aa.slice(2,-2)); //[3,4] 相对于倒数第二个之前  console.log(aa.slice(3)); //[4,5,6] 一个参数从第三个到最后  console.log(aa.slice(-2));//[5,6] 一个参数负值从倒数第二个到最后 |

### 3.4.6、push()

在数组末尾添加一个或多个元素，并返回新数组长度

### 3.4.7、pop()

从数组末尾删除1个元素(删且只删除1个), 并返回 被删除的元素

### 3.4.8、shift()

在数组开始添加一个或多个元素，并返回新数组长度

### 3.4.9、unshift()

在数组开始删除一个元素(删且只删除1个),并返回 被删除的元素

### 3.4.10、toString()和toLocaleString()

将数组的每个元素转化为字符串，并且输入用逗号分隔的字符串列表。功能类似join();

## 3.5、数组中的高阶函数

### 3.5.1、sort()

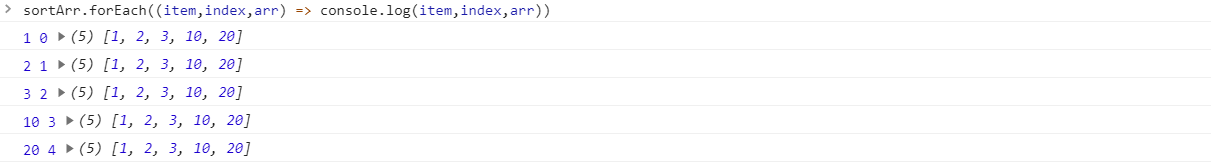
默认情况下sort()方法没有传比较函数的话，默认按字母升序，如果不是元素不是字符串的话，会调用toString()方法将元素转化为字符串的Unicode(万国码)位点，然后再比较字符。所以用默认方法排序数据是有问题的。

|  |
| --- |
| var arr = [20,10,2,1,3];  arr.sort();// [1, 10, 2, 20, 3]  arr.sort(function(a,b){  return a-b; //升序  }); //[1, 2, 3, 10, 20]  arr.sort(function(a,b){  return b-a; //降序  }); //[20,10,3,2,1] |

### 3.5.2、forEach()

从头至尾遍历数组，为每个元素调用指定函数

输入为一个待遍历函数，函数的参数依次为：数组元素、元素的索引、数组本身



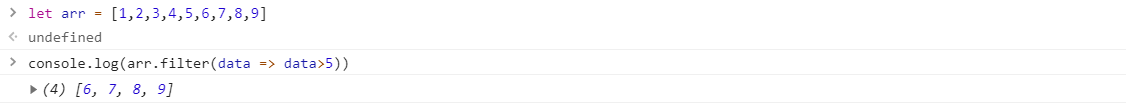
### 3.5.3、map()

调用的数组的每一个元素传递给指定的函数，并返回一个新数组 ,不修改原数组。

|  |
| --- |
| var arr = [2,3,4,5,6];  var bb= arr.map(function(x){  return x\*x;  });  console.log(bb); // [4, 9, 16, 25, 36] |

### 3.5.4、filter()

根据传入函数的返回值true和false来判断是否留下或过滤掉



### 3.5.5、every()和some()

every() 判断数组中每一项都是否满足条件，只有所有项都满足条件，才会返回true。

some() 判断数组中是否存在满足条件的项，只要有一项满足条件，就会返回true。

|  |
| --- |
| var arr = [1, 2, 3, 4, 5, 6];  arr.every(x=>x>0);//return true;  arr.every(x=>x>5);//return false;  arr.some(x=>x>5);//return true; |

### 3.5.6、 reduce()和reduceRight()

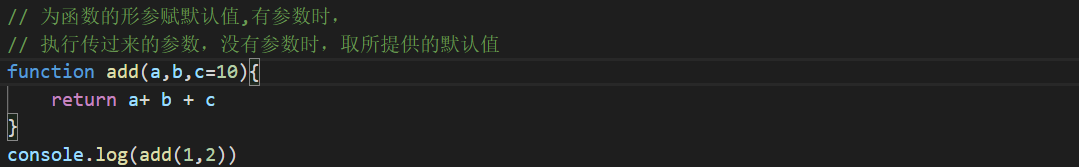
reduce()两个参数：函数和递归的初始值。从数组的第一项开始，逐个遍历到最后

reduceRight()从数组的最后一项开始，向前遍历到第一项

|  |
| --- |
| //可以用reduce快速求数组之和  var arr=[1,2,3,4];  arr.reduce(function(a,b){  return a+b;  }); //10 |

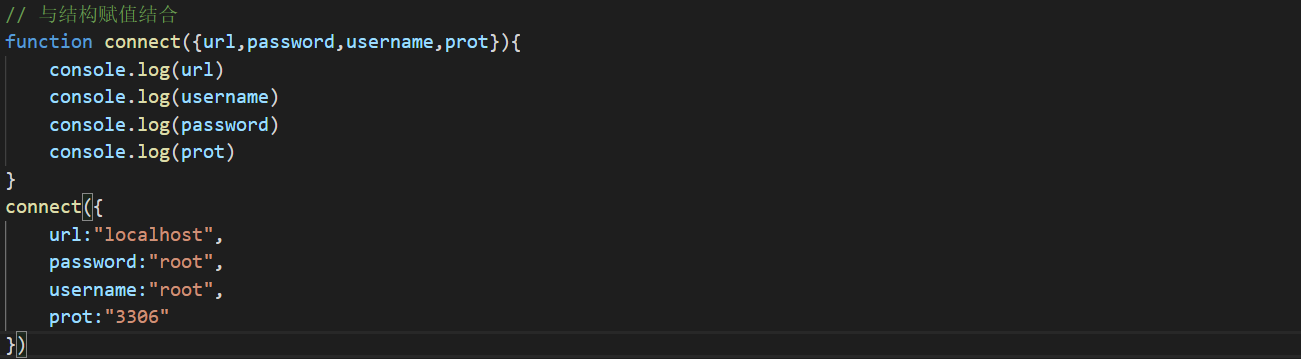
## 3.6、项目开发中常用的ES6语法

### 3.6.1、形参的默认赋值



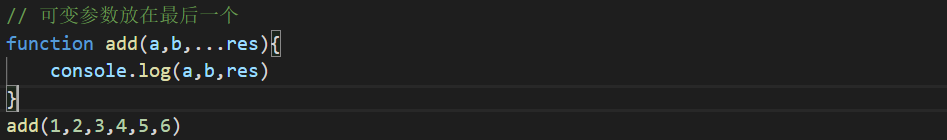
结果是13

### 3.6.2、形参与解构赋值结合



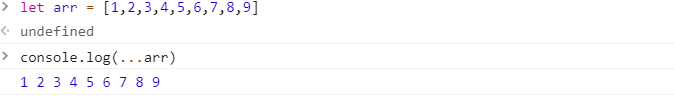


### 3.6.3、可变参数





### 3.6.4、扩展运算符



### 3.6.5、promise语法

**3.6.5.1、基本使用**

|  |
| --- |
|  |

**3.6.5.2、then方法**

|  |
| --- |
| then方法的返回结果是Promise 对象, 对象状态由回调函数的执行结果决定  1. 如果回调函数中返回的结果是 非 promise 类型的属性, 状态为成功, 返回值为对象的成功的值        1.1、如果then方法无返回值，即非promise类型的属性，状态为成功，返回值为undefined        2、返回值是promise对象，then方法返回结果与返回的promise对象的状态有关，返回的promise对象的状态为正确的，then方法返回的promise对象的状态为正确的；返回的promise对象的状态为错误的，then方法返回的promise对象的状态为错误的  **正确状态**        返回结果为错误的        返回值抛出一个异常对象      返回值抛出字符串 |

**3.6.5.3、catch方法**

|  |
| --- |
| catch用于捕获Promise对象抛出的异常，相当于省略第一个函数参数的then方法 |

**3.6.5.4、all方法和race方法**

|  |
| --- |
| Promise.all([p1,p2,p3]).then(res => {  }).catch(res => {  })  成功的时候返回的是一个结果数组，而失败的时候则返回最先被reject失败状态的值。  **成功状态**        **失败状态**        Promise.all获得的成功结果的数组里面的数据顺序和Promise.all接收到的数组顺序是一致的，即p1的结果在前，即便p1的结果获取的比p2要晚        Promise.race([p1, p2, p3])里面哪个结果获得的快，就返回那个结果，不管结果本身是成功状态还是失败状态。 |

### 3.6.6、export暴露语法

|  |
| --- |
| 1、单一暴露      2、统一暴露      3、默认暴露 |

### 3.6.7、import倒入语法

|  |
| --- |
|  |

### 3.6.8、项目常用的async函数

|  |
| --- |
| async函数    1、返回类型为非Promise对象，async函数返回的结果是成功的promise      2、返回类型为异常，async函数返回的结果是失败的promise      3、返回promise类型的值，async函数返回的promise对象的状态由返回的promise状态决定  **返回的为成功的Promise**        **返回的为失败的Promise** |

### 3.6.9、项目中常用的await表达书

|  |
| --- |
| **await后面返回的成功的值**        **await后面返回的失败的值** |

### 3.6.10、async和await获得ajax请求获得数据

|  |
| --- |
|  |

# 4、项目使用的使用的组件框架--View UI

组件框架的地址

<https://iviewui.com/docs/introduce>

## 4.1、组件框架的使用(以分页组件page为例)

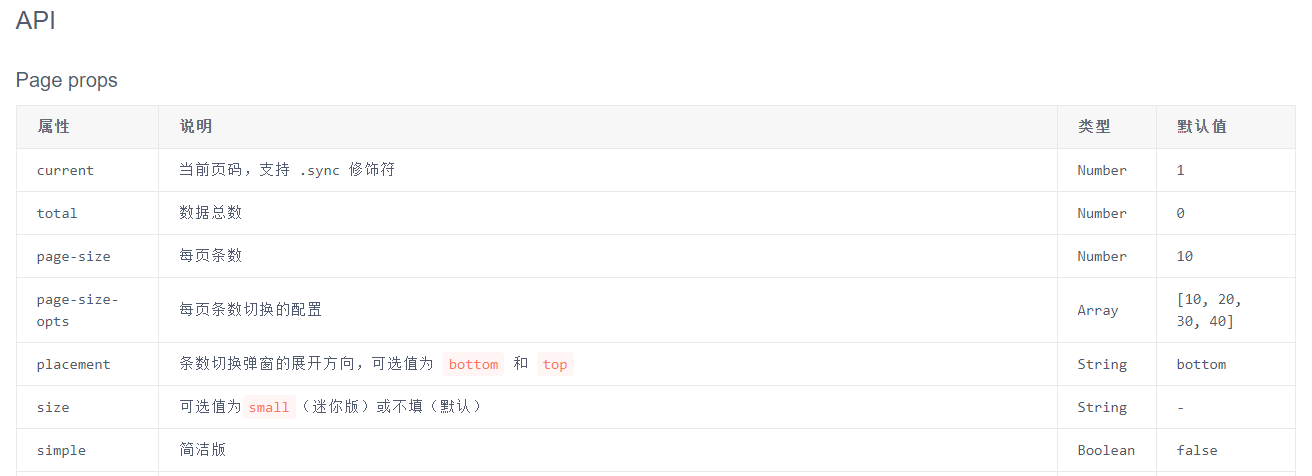
由于本项目已经全局引入了该组件框架，因此，在需要使用的直接按照文档使用即可。

左侧为组件的页面展示情况，右侧为使用的代码



## 4.2、组件参数api

组件通过这些api属性来控制组件的



## 4.3、组件触发的事件

组件使用点击时触发的事件



## 4.4、组件的插槽使用

注意：插槽的使用方式具体上文有消息介绍

