

-

整理自互联网，仅供学习交流。



2018-10-30

[公司名称]

[公司地址]

目录

[一、 Vue.js 与Webpack 1](#_Toc5094763)

[1.1 vue.js 研究 1](#_Toc5094764)

[1.2 vue.js 介绍 1](#_Toc5094765)

[1.2.1 vue.js是什么？ 1](#_Toc5094766)

[1.2.1 Vue.js与ECMAScript 1](#_Toc5094767)

[1.2.2 Vue.js的使用 1](#_Toc5094768)

[1.2.3 vue.js有哪些功能？ 2](#_Toc5094769)

[1.3 vue.js 基础 2](#_Toc5094770)

[1.3.1 MVVM模式 2](#_Toc5094771)

[1.3.2 入门程序 3](#_Toc5094772)

[1.3.3 代码编写步骤： 3](#_Toc5094773)

[1.3.4 1+1=2 4](#_Toc5094774)

[1.3.5 v-if 和v-for 6](#_Toc5094775)

[1.4 webpack 入门 7](#_Toc5094776)

[1.4.1 安装webpack 8](#_Toc5094777)

[1.4.2 安装webpack 12](#_Toc5094778)

[1.4.3 入门程序 12](#_Toc5094779)

[1.4.4 webpack-dev-server 15](#_Toc5094780)

[二、 常见问题 19](#_Toc5094781)

[2.1 webstorm无法编辑代码问题 19](#_Toc5094782)

[2.2 解决npm一直停在"checking installable status"的问题 20](#_Toc5094783)

[2.3 cnpm安装失败 20](#_Toc5094784)

[2.3.1 安装淘宝镜像： 20](#_Toc5094785)

[2.4 解决cnpm不是外部命令： 21](#_Toc5094786)

[2.5 webpack失败 23](#_Toc5094787)

[2.6 Webpack4 打包出错问题解决 24](#_Toc5094788)

# Vue.js 与Webpack

## vue.js 研究

## vue.js 介绍

### vue.js是什么？

Vue (读音 /vjuː/，类似于 view) 是一套用于构建用户界面的渐进式框架。与其它大型框架不同的是，Vue 被设计

为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。另一

方面，当与现代化的工具链以及各种 支持类库结合使用时， Vue 也完全能够为复杂的单页应用提供驱动。

渐进式框架：Progressive，说明vue.js的轻量，是指一个前端项目可以使用vue.js一两个特性也可以整个项目都用

vue.js。

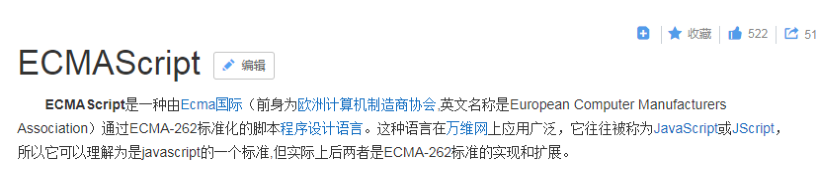
自底向上逐层应用：作为渐进式框架要实现的目标就是方便项目增量开发。

参考：<https://cn.vuejs.org/v2/guide/>

### Vue.js与ECMAScript

Vue 不支持 IE8 及以下版本，因为 Vue 使用了 IE8 无法模拟的 ECMAScript 5 特性。

什么是ECMAScript?



ECMAScript（简称ES）是一种规范，我们平常所说的Js/Javascript是ECMAScript的实现，早期主要应用的ES3，当

前主流浏览器都支持ES5、ES6，ES8已于2017年发布。

ES6：http://www.ecma-international.org/ecma-262/6.0/

ES7 ：http://www.ecma-international.org/ecma-262/7.0/

### Vue.js的使用

1）在html页面使用script引入vue.js的库即可使用。

2）使用Npm管理依赖，使用webpack打包工具对vue.js应用打包。

大型应用推荐此方案。

3）Vue-CLI脚手架

使用 vue.js官方提供的CLI脚本架很方便去创建vue.js工程雏形。

### vue.js有哪些功能？

1）声明式渲染

Vue.js 的核心是一个允许采用简洁的模板语法来声明式地将数据渲染进 DOM 的系统。

比如：使用vue.js的插值表达式放在Dom的任意地方， 差值表达式的值将被渲染在Dom中。

2）条件与循环

dom中可以使用vue.js提供的v-if、v-for等标签，方便对数据进行判断、循环。

3）双向数据绑定

Vue 提供v-model 指令，它可以轻松实现Dom元素和数据对象之间双向绑定，即修改Dom元素中的值自动修改绑

定的数据对象，修改数据对象的值自动修改Dom元素中的值。

4）处理用户输入

为了让用户和你的应用进行交互，我们可以用 v -on 指令添加一个事件监听器，通过它调用在 Vue 实例中定义的

方法

5）组件化应用构建

vue.js可以定义一个一个的组件，在vue页面中引用组件，这个功能非常适合构建大型应用。

## vue.js 基础

### MVVM模式

vue.js是一个MVVM的框架，理解MVVM有利于学习vue.js。

MVVM 拆分解释为：

Model: 负责数据存储

View: 负责页面展示

View Model: 负责业务逻辑处理（比如Ajax请求等），对数据进行加工后交给视图展示

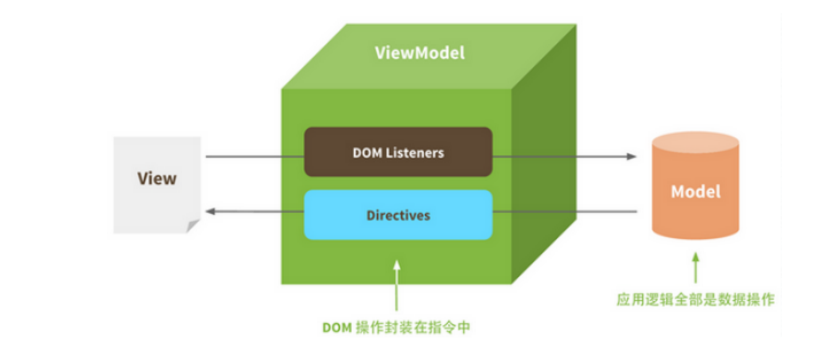
MVVM 要解决的问题是将业务逻辑代码与视图代码进行完全分离，使各自的职责更加清晰，后期代码维护更

加简单

用图解的形式分析 Ajax请求回来数据后直接操作Dom来达到视图的更新的缺点，以及使用MVVM模式是如何

来解决这个缺点的

Vue 中的 MVVM



从上图看出，VM(ViewModel)可以把view视图和Model模型解耦合，VM的要做的工作就是vue.js所承担的。

v-on，用@缩写，v-bind用:缩写

### 入门程序

本次测试我们在门户目录中创建一个html页面进行测试，正式的页面管理前端程序会单独创建工程。

在门户目录中创建vuetest目录，并且在目录下创建vue\_01.html文件

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <title>vue.js测试程序</title>  <script src="vue.min.js"></script>  </head>  <body>  <!--实现在body区域显示一个名称-->  <div id="app">  {{name}}<!--相当于MVVM中的view视图-->  </div>  </body>  <script>  //编写MVVM中的model部分以及V(ViewModel)部分  var VM = new Vue({  el:'#app',//vm接管了app区域的管理  data:{//model数据  name:'test'  }  });  </script>  </html> |

### 代码编写步骤：

1、定义html，引入vue.js

2、定义app div，此区域作为vue的接管区域

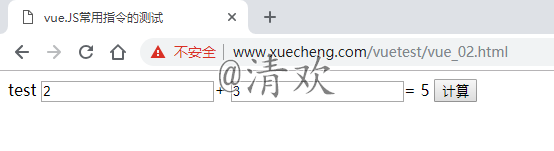
3、定义vue实例，接管app区域。

4、定义model（数据对象）

5、VM完成在app中展示数据

### 1+1=2

实现效果：



|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <title>vue.JS常用指令的测试</title>  <script src="vue.min.js"></script>  </head>  <body>  <!--实现在body区域显示一个名称-->  <div id="app">  {{name}}<!--相当于MVVM中的view视图-->  <input type="text" v-model="num1"/>+  <input type="text" v-model="num2"/>=  {{Number.parseInt(num1)+Number.parseInt(num2)}}  <button>计算</button>  </div>  </body>  <script>  //编写MVVM中的model部分以及V(ViewModel)部分  var VM = new Vue({  el:'#app',//vm接管了app区域的管理  data:{//model数据  name:'test',  num1:2,  num2:2  }  });  </script>  </html> |

本例子学习了：

#### v-model：

1、在表单控件或者组件上创建双向绑定 2、v-model仅能在如下元素中使用：

input

select

textarea

components（Vue中的组件）

#### v-text

解决插值表达式闪烁问题，使用v-text

v-text可以将一个变量的值渲染到指定的元素中,它可以解决插值表达式闪烁的问题

|  |
| --- |
| <div id="app">  <!--{{name}}&lt;!&ndash;相当于MVVM中的view视图&ndash;&gt;-->  <span v-text="name"></span>  <input type="text" v-model="num1"/>+  <input type="text" v-model="num2"/>=  <!--{{Number.parseInt(num1)+Number.parseInt(num2)}}-->  <span v-text="Number.parseInt(num1)+Number.parseInt(num2)"></span>  <button>计算</button>  </div> |

#### v-on

绑定一个按钮的单击事件

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <title>vue.JS常用指令的测试</title>  <script src="vue.min.js"></script>  </head>  <body>  <!--实现在body区域显示一个名称-->  <div id="app">  <!--{{name}}&lt;!&ndash;相当于MVVM中的view视图&ndash;&gt;-->  <span v-text="name"></span>  <input type="text" v-model="num1"/>+  <input type="text" v-model="num2"/>=  <!--{{Number.parseInt(num1)+Number.parseInt(num2)}}-->  <!--<span v-text="Number.parseInt(num1)+Number.parseInt(num2)"></span>-->  <span v-text="result"></span>  <button v-on:click="change">计算</button>  </div>  </body>  <script>  //编写MVVM中的model部分以及V(ViewModel)部分  var VM = new Vue({  el:'#app',//vm接管了app区域的管理  data:{//model数据  name:'test',  num1:0,  num2:0,  result:0  },  methods:{  change:function () {  this.result=Number.parseInt(this.num1)+Number.parseInt(this.num2)  // alert("计算结果"+this.result)  }  }  });  </script>  </html> |

#### v-bind

1、作用：

v‐bind可以将数据对象绑定在dom的任意属性中。

v‐bind可以给dom对象绑定一个或多个特性，例如动态绑定style和class

2、举例：

<img v‐bind:src="imageSrc">

<div v‐bind:style="{ fontSize: size + 'px' }"></div>

3、缩写形式

<img :src="imageSrc">

<div :style="{ fontSize: size + 'px' }"></div>

### v-if 和v-for

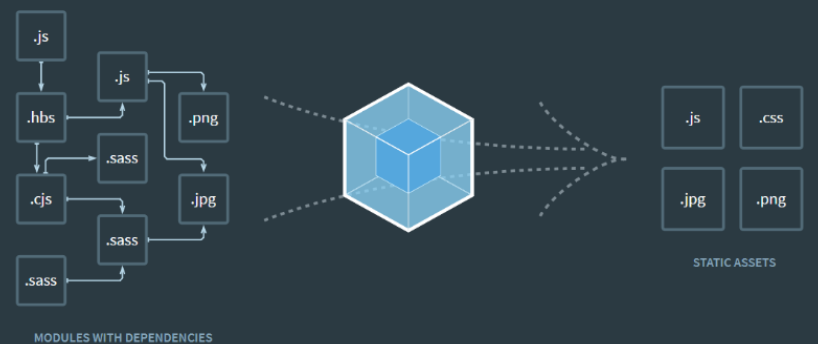
|  |
| --- |
| <!DOCTYPE html>  <html lang="en" xmlns:v-bind="http://www.w3.org/1999/xhtml" xmlns:v-on="http://www.w3.org/1999/xhtml">  <head>  <meta charset="UTF-8">  <title>vue.JS常用指令的测试</title>  <script src="vue.min.js"></script>  </head>  <body>  <!--实现在body区域显示一个名称-->  <div id="app">  <ul>  <li v-for="(item,index) in list" :key="index" v-if="index % 2 == 0 ">{{index}}---{{item}}</li>  <li v-for="(value,key) in user">{{key}}----{{value}}</li>  <li v-for="(item,index) in userlist" :key="item.user.username">  <div v-if="item.user.username =='test1'" style="background:#00f50c">  {{index}}---{{item.user.username}}---{{item.user.age}}  </div>  <div v-else>  {{index}}---{{item.user.username}}---{{item.user.age}}  </div>  </li>  </ul>  </div>  </body>  <script>  //编写MVVM中的model部分以及V(ViewModel)部分  var VM = new Vue({  el: '#app',//vm接管了app区域的管理  data: {//model数据  list: [1, 2, 3, 4, 5, 6],  user: {username: 'test', age: 10},  userlist:[  { user: {username: 'test1', age: 11},},  { user: {username: 'test2', age: 12},}  ]  }  });  </script>  </html> |

## webpack 入门

使用vue.js开发大型应用需要使用webpack打包工具，本节研究webpack的使用方法。

webpack介绍

Webpack 是一个前端资源的打包工具，它可以将js、image、css等资源当成一个模块进行打包。



从图中我们可以看出，Webpack 可以将js、css、png等多种静态资源 进行打包，使用webpack有什么好处呢？

1、模块化开发

程序员在开发时可以分模块创建不同的js、 css等小文件方便开发，最后使用webpack将这些小文件打包成一个文

件，减少了http的请求次数。

webpack可以实现按需打包，为了避免出现打包文件过大可以打包成多个文件。

2、 编译typescript、ES6等高级js语法

随着前端技术的强大，开发中可以使用javascript的很多高级版本，比如：typescript、ES6等，方便开发，

webpack可以将打包文件转换成浏览器可识别的js语法。

3、CSS预编译

webpack允许在开发中使用Sass 和 Less等原生CSS的扩展技术，通过sass-loader、less-loader将Sass 和 Less的

语法编译成浏览器可识别的css语法。

webpack的缺点：

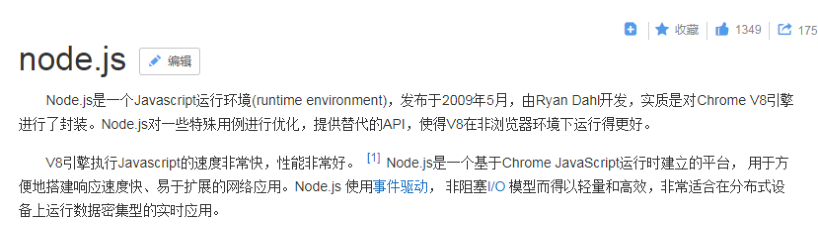
1、配置有些繁琐

2、文档不丰富

### 安装webpack

#### 安装Node.js

webpack基于node.js运行，首先需要安装node.js。



为什么会有node.js？

传统意义上的 JavaScript 运行在浏览器上，Chrome 使用的 JavaScript 引擎是 V8，Node.js 是一个运行在服务端

的框架，它的底层就使用了 V8 引擎，这样就可以使用javascript去编写一些服务端的程序，这样也就实现了用

javaScript去开发 Apache + PHP 以及 Java Servlet所开发的服务端功能，这样做的好处就是前端和后端都采用

javascript，即开发一份js程序即可以运行在前端也可以运行的服务端，这样比一个应用使用多种语言在开发效率上

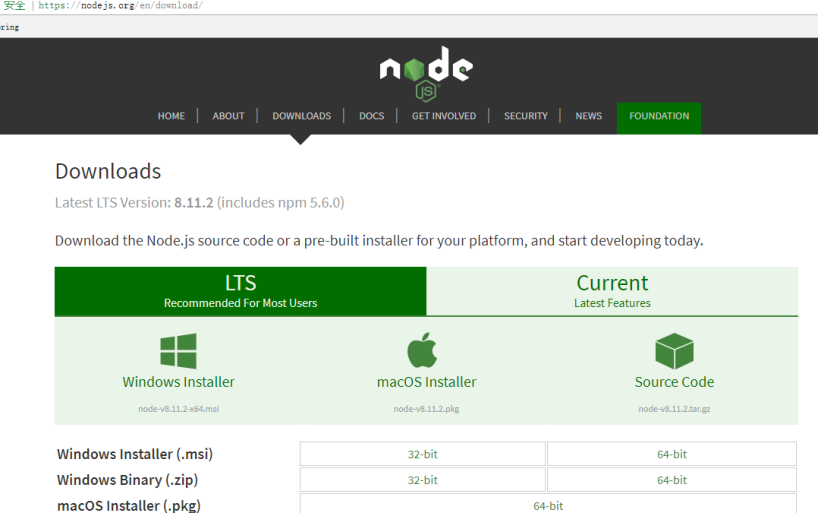
要高，不过node.js属于新兴产品，一些公司也在尝试使用node.js完成一些业务领域，node.js基于V8引擎，基于

事件驱动机制，在特定领域性能出色，比如用node.js实现消息推送、状态监控等的业务功能非常合适。

下边我们去安装Node.js：

1、下载对应你系统的Node.js版本:

<https://nodejs.org/en/download/>



推荐下载LTS版本，本教程安装9.4.0。

2 、选安装目录进行安装

默认即可

安装完成检查PATH环境变量是否设置了node.js的路径。

3、测试

在命令提示符下输入命令

node -v

会显示当前node的版本

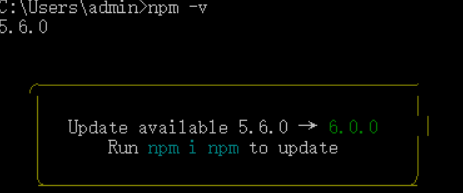
#### 安装NPM

1、自动安装NPM

npm全称Node Package Manager，他是node包管理和分发的工具，使用NPM可以对应用的依赖进行管理，NPM

的功能和服务端项目构建工具maven差不多，我们通过npm 可以很方便地下载js库，打包js文件。

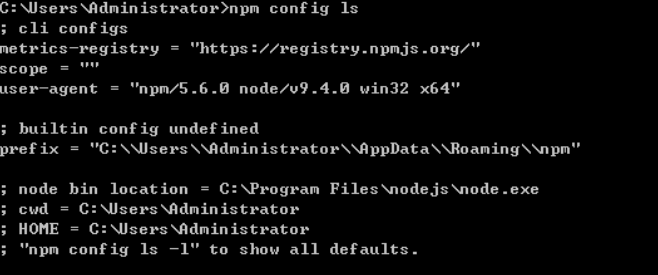
node.js已经集成了npm工具，在命令提示符输入 npm -v 可查看当前npm版本



2、设置包路径

包路径就是npm从远程下载的js包所存放的路径。

使用 npm config ls 查询NPM管理包路径（NPM下载的依赖包所存放的路径）



NPM 默认的管理包路径在C:/用户/[用户名]/AppData/Roming/npm/node\_meodules，为了方便对依赖包管理，我

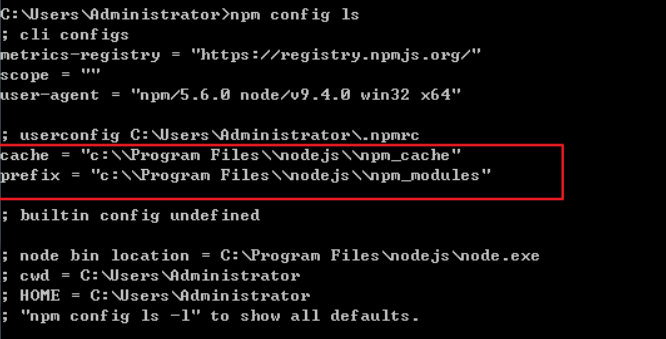
们将管理包的路径设置在单独的地方，本教程将安装目录设置在node.js的目录下，创建npm\_modules和npm\_cache，执行下边的命令：

本教程安装node.js在D:\Program Files\nodejs下所以执行命令如下：

npm config set prefix "C:\Program Files\nodejs\npm\_modules"

npm config set cache "c:\Program Files\nodejs\npm\_cache"

此时再使用 npm config ls 查询NPM管理包路径发现路径已更改



3、安装cnpm

npm默认会去国外的镜像去下载js包，在开发中通常我们使用国内镜像，这里我们使用淘宝镜像

下边我们来安装cnpm：

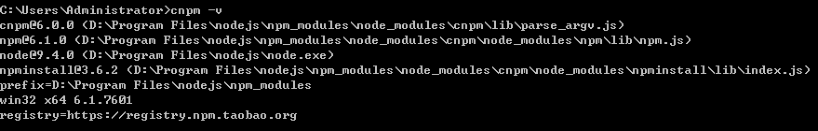
有时我们使用npm下载资源会很慢，所以我们可以安装一个cnmp(淘宝镜像)来加快下载速度。

输入命令，进行全局安装淘宝镜像。

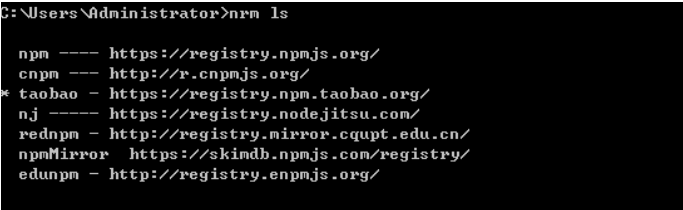
npm install -g cnpm --registry=https://registry.npm.taobao.org

安装后，我们可以使用以下命令来查看cnpm的版本

cnpm –v



nrm ls 查看镜像已经指向taobao



使nrm use XXX切换 镜像

如果nrm没有安装则需要进行全局安装：cnpm install -g nrm

4、非连网环境安装cnpm

从本小节第3步开始就需要连网下载npm包，如果你的环境不能连网在老师的资料文件下有已经下载好的webpack相关包，下边是安装方法。

1）配置环境变量

NODE\_HOME = D:\Program Files\nodejs (node.js安装目录)

在PATH变量中添加：%NODE\_HOME%;%NODE\_HOME%\npm\_modules;

2）找到npm包路径

根据上边的安装说明npm包路径被设置到了node.js安装目录下的npm\_modules目录。

可以使用npm config ls查看。

拷贝课程资料中的 npm\_modules.zip到node.js安装目录，并解压npm\_modules.zip覆盖本目录下的

npm\_modules文件夹。

3）完成上边步骤测试

cnpm -v

### 安装webpack

1、连网安装

webpack安装分为本地安装和全局安装：

本地安装：仅将webpack安装在当前项目的node\_modules目录中，仅对当前项目有效。

全局安装：将webpack安装在本机，对所有项目有效，全局安装会锁定一个webpack版本，该版本可能不适用某个项目。全局安装需要添加 -g 参数。

进入webpacktest测试目录目录，运行：

#### 本地安装：

只在我的项目中使用 webpack，需要进行本地安装，因为项目和项目所用的webpack的版本不一样。本地安装就会

将webpack的js包下载到项目下的npm\_modeuls目录下。

在门户目录下创建webpack测试目录webpacktest01：

npm install --save-dev webpack 或 cnpm install --save-dev webpack

npm install --save-dev webpack-cli (4.0以后的版本需要安装webpack-cli)

2）全局安装加-g，如下：

全局安装就将webpack的js包下载到npm的包路径下。

npm install webpack -g 或 cnpm install webpack -g

3）安装webpack指定的版本：

本教程使用webpack3.6.0，安装webpack3.6.0：

进入webpacktest测试目录，运行：cnpm install --save-dev webpack@3.6.0

全局安装：npm install webpack@3.6.0 -g或 cnpm install webpack@3.6.0 -g

#### 非连网安装

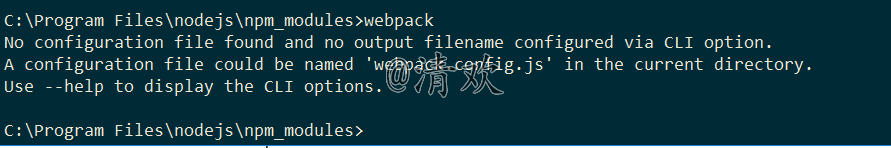
参考上边 “非连网环境安装cnpm”描述，将课程资料中的 npm\_modules.zip到node.js安装目录，并解压

npm\_modules.zip覆盖本目录下的npm\_modules文件夹。

说明：已执行 “非连网环境安装cnpm”下的操作不用重复执行。

测试：

在cmd状态输入webpack，出现如下提示说明 webpack安装成功



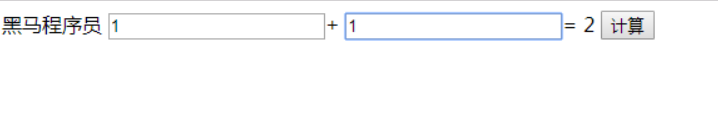
### 入门程序

通过本入门程序体会webpack打包的过程及模块化开发的思想。

1.3.3.1 需求分析

通过入门程序实现对js文件的打包，体会webpack是如何对应用进行模块化管理。

对上边1+1=2的例子使用webpack进行模块化管理



#### 定义模块

创建webpacktest01目录，将vue.min.js及vue\_02.html拷贝到目录下。

1、定义model01.js

在webpacktest01目录下创建model01.js

将本程序使用的加法运算的js方法抽取到一个js文件，此文件就是一个模块

|  |
| --- |
| // 定义add函数  function add(x, y) {      return x + y  }  // function add2(x, y) {  //     return x + y+1  // }  // 导出add方法  module.exports.add = add;  // module.exports ={add,add2};//如果有多个方法这样导出  // module.exports.add2 = add2//如果有多个方法也可以这样导出 |

2、定义main.js

在webpacktest01目录下创建main.js，main.js是本程序的js主文件，包括如下内容：

1、在此文件中会引用model01.js模块

2、引用vue.min.js（它也一个模块）

3、将html页面中构建vue实例的代码放在main.js中。

main.js的代码如下

|  |
| --- |
| var {add} = require('./model01.js');  var Vue = require('./vue.min');  var VM = new Vue({      el:"#app",//表示当前vue对象接管app的div区域      data:{          name:'黑马程序员',// 相当于是MVVM中的Model这个角色          num1:0,          num2:0,          result:0,          url:'http://www.itcast.cn'      },      methods:{          change:function(){           //这里使用了导入的model01.js文件中的add方法              this.result = add(Number.parseInt(this.num1),Number.parseInt(this.num2))              alert(this.result)          }  }  }); |

打包测试

上边将mode01.js模块及main.js主文件编写完成，下边使用webpack对这些js文件进行打包

1、进入程序目录，执行webpack main.js build.js ，这段指令表示将main.js打包输出为 build.js文件

执行完成，观察程序目录是否出现build.js。

2、在html中引用build.js

|  |
| --- |
| <!DOCTYPE html>  <html lang="en" xmlns:v‐on="http://www.w3.org/1999/xhtml">  <head>      <meta charset="UTF‐8">      <title>vue.js入门程序</title>  </head>  <body>  <div id="app">      <!‐‐{{name}}解决闪烁问题使用v‐text‐‐>  <a v‐bind:href="url"><span v‐text="name"></span></a>  <input type="text" v‐model="num1">+  <input type="text" v‐model="num2">=  <span v‐text="result"></span>      <!‐‐{{num1+num2}}‐‐>  <!‐‐<input type="text" v‐model="result">‐‐>      <button v‐on:click="change">计算</button>      <!‐‐ 在Vue接管区域中使用Vue的系统指令呈现数据      这些指令就相当于是MVVM中的View这个角色 ‐‐>  </div>  </body>  <script src="build.js"></script>  <script>  </script>  </html> |

3、测试总结

测试功能：

1）输入任意加数，其和会自动计算

2）点击“计算”会调用model01.js中的add方法

总结：

webpack可以将js分模块开发，开发完成对各模块代码打包成一个统一的文件。

前端模块开发的思想和服务端模块开发的思想是一致的，有利于多人协助开发。

### webpack-dev-server

webpack-dev-server开发服务器，它的功能可以实现热加载 并且自动刷新浏览器。

创建一个新的程序目录，这里我们创建webpacktest02目录，将webpack入门程序的代码拷贝进来，并在目录下创

建src目录、dist目录。

将main.js和model01.js拷贝到src目录。

#### 安装配置

1、安装webpack-dev-server

使用 webpack-dev-server需要安装webpack、 webpack-dev-server和 html-webpack-plugin三个包。

cnpm install webpack@3.6.0 webpack-dev-server@2.9.1 html-webpack-plugin@2.30.1 --save-dev

安装完成，会发现程序目录出现一个package.json文件，此文件中记录了程序的依赖。

没有联网的同学拷贝老师提供的node\_modules.zip到webpacktest02目录下，解压到node\_modules目录下。

2、配置webpack-dev-server

在package.json中配置script

|  |
| --- |
| "scripts": {  "dev": "webpack-dev-server --inline --hot --open --port 5008"  }, |

--inline：自动刷新

--hot：热加载

--port：指定端口

--open：自动在默认浏览器打开

--host：可以指定服务器的 ip，不指定则为127.0.0.1，如果对外发布则填写公网ip地址

此时package.json的文件内容如下：

|  |
| --- |
| {  "scripts": {  "dev": "webpack-dev-server --inline --hot --open --port 5008"  },  "devDependencies": {  "html-webpack-plugin": "^2.30.1",  "webpack": "^3.6.0",  "webpack-dev-server": "^2.9.1"  }  } |

devDependencies：开发人员在开发过程中所需要的依赖。

scripts：可执行的命令

#### 配置webpack.config.js

在webpacktest02目录下创建 webpack.config.js， webpack.config.js是webpack的配置文件。在此文件中可以配置应用的入口文件、输出配置、插件等，其中要实现热加载自动刷新功能需要配置html-webpack-plugin插件。

html-webpack-plugin的作用是根据html模板在内存生成html文件，它的工作原理是根据模板文件在内存中生成一个index.html文件。

1、配置模板文件

将原来的vue\_02.html作为模板文件，为了和内存中的index.html文件名区别，注意将vue\_02.html中的script标签去掉，内容如下：

|  |
| --- |
| <!DOCTYPE html>  <html lang="en" xmlns:v‐on="http://www.w3.org/1999/xhtml">  <head>      <meta charset="UTF‐8">      <title>vue.js入门程序</title>  </head>  <body>  <div id="app">      <!‐‐{{name}}解决闪烁问题使用v‐text‐‐>  <a v‐bind:href="url"><span v‐text="name"></span></a>  <input type="text" v‐model="num1">+  <input type="text" v‐model="num2">=  <span v‐text="result"></span>      <!‐‐{{num1+num2}}‐‐>  <!‐‐<input type="text" v‐model="result">‐‐>      <button v‐on:click="change">计算</button>      <!‐‐ 在Vue接管区域中使用Vue的系统指令呈现数据      这些指令就相当于是MVVM中的View这个角色 ‐‐>  </div>  </body>  </html> |

#### 配置 html-webpack-plugin

在webpack.config.js中配置html-webpack-plugin插件

|  |
| --- |
| var htmlwp = require('html‐webpack‐plugin');  module.exports={      entry:'./src/main.js',  //指定打包的入口文件      output:{          path : \_\_dirname+'/dist',  // 注意：\_\_dirname表示webpack.config.js所在目录的绝对路径          filename:'build.js'    //输出文件      },      plugins:[          new htmlwp({              title: '首页',  //生成的页面标题<head><title>首页</title></head>              filename: 'index.html', //webpack‐dev‐server在内存中生成的文件名称，自动将build注入到这  个页面底部，才能实现自动刷新功能              template: 'vue\_02.html' //根据index1.html这个模板来生成(这个文件请程序员自己生成)          })      ]  } |

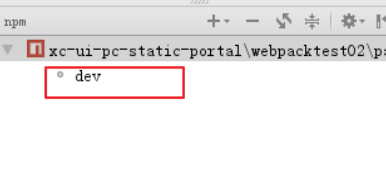
启动

启动文件：

1、进入 webpacktest02目录，执行npm run dev

2、使用webstorm，右键package.json文件，选择“Show npm Scripts”

打开窗口：



双击 dev。

注意：dev就是在package.json中配置的webpack dev server命令。

发现启动成功自动打开浏览器。

修改src中的任意文件内容，自动加载并刷新浏览器。

debug调试

使用了webpack之后就不能采用传统js的调试方法在chrome中打断点。

webpack 将多个源文件打包成一个文件，并且文件的内容产生了很大的变化，webpack提供devtool进行调试，

devtool是基于sourcemap的方式，在调试时会生成一个map文件，其内容记录生成文件和源文件的内容映射，即

生成文件中的哪个位置对应源文件中的哪个位置，有了sourcemap就可以在调试时看到源代码。

配置如下：

1、在webpack.config.js中配置：

devtool: 'eval-source-map',

webpack.config.js部分内容如下：

|  |
| --- |
| var htmlwp = require('html-webpack-plugin');  module.exports =  {  entry: './src/main.js', //指定打包的入口文件  output:  {  path: \_\_dirname + '/dist', // 注意：\_\_dirname表示webpack.config.js所在目录的绝对路径  filename: 'build.js' //输出文件  },  devtool: 'eval-source-map',  plugins: [  new htmlwp(  {  title: '首页', //生成的页面标题<head><title>首页</title></head>  filename: 'index.html', //webpack‐dev‐server在内存中生成的文件名称，自动将build注入到这个页面底部，才能实现自动刷新功能  template: 'vue\_02.html' //根据index1.html这个模板来生成(这个文件请程序员自己生成)  }  )  ]  } |

2、在js中跟踪代码的位置上添加debugger

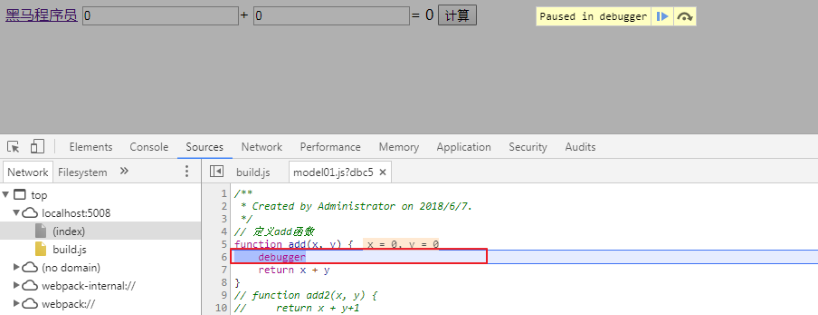
一个例子：

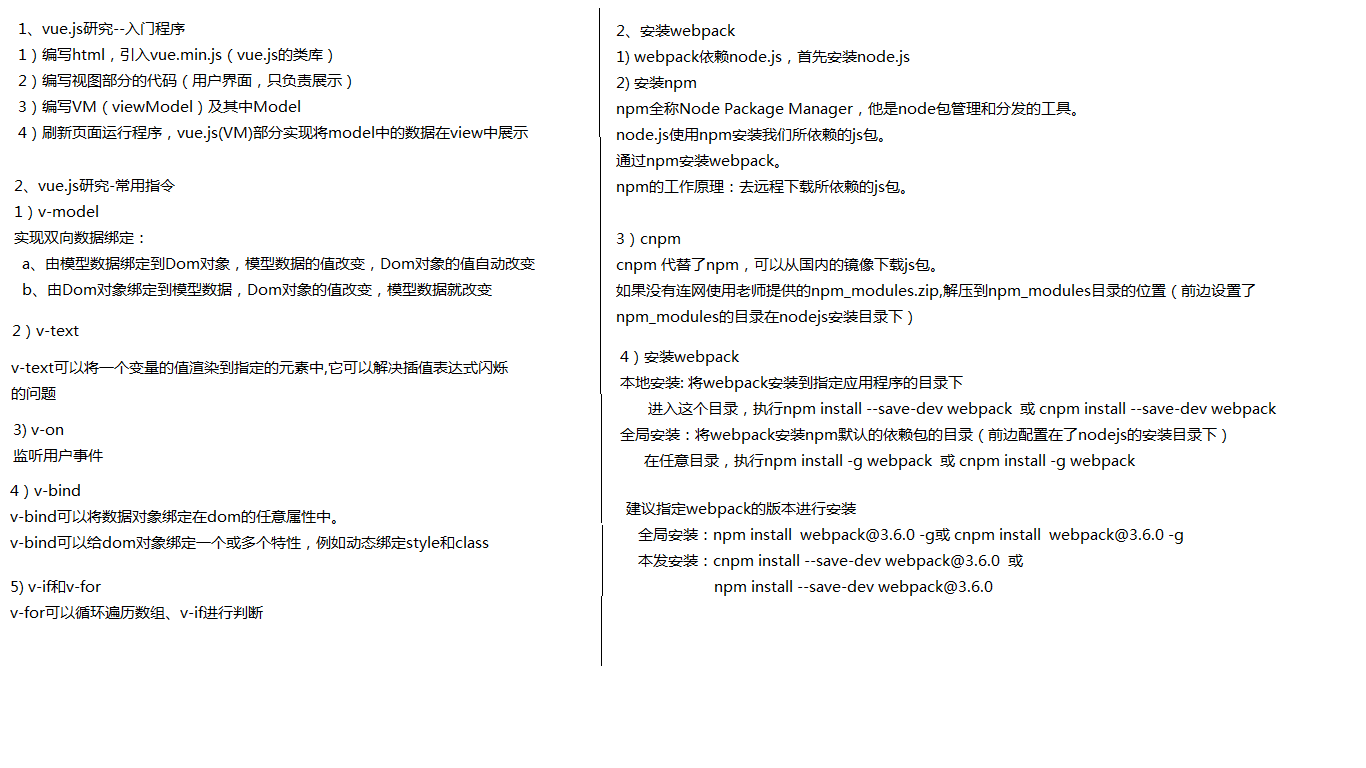
在add方法中添加debugger

|  |
| --- |
| // 定义add函数  function add(x, y) {      debugger      return x + y  } |

启动应用，刷新页面跟踪代码：

点击“计算” 即进入debugger代码位置，此时可以使用chrome进行调试了。



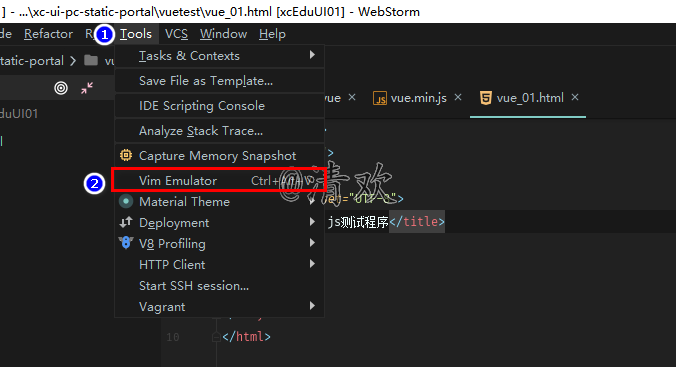




# 常见问题

## webstorm无法编辑代码问题

方法：工具栏Tools -> Vim Emulator 选项取消勾选



## 解决npm一直停在"checking installable status"的问题

原因：

旧的npm缓存与项目冲突

解决方法：

步骤一

sudo npm cache clean

步骤二

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

## cnpm安装失败

### 安装淘宝镜像：

要安装Angular4。于是我对着一股浓郁口音的视频开启了Angular4安装之旅。那口音说了，ang哥乐4不是那么好装的，由于我国的墙，所以我们要通过淘宝的镜像去安装。

于是先安装淘宝镜像，cnpm。这个百度一大堆。

npm install -g cnpm --registry=https://registry.npm.taobao.org

安装完了之后检查是否安装成功。

cnpm -v

若果出现以下东西说明安装成功。

C:\Users\WenCh> cnpm -v

cnpm@5.1.1 (F:\Live\NODE\node\_global\node\_modules\cnpm\lib\parse\_argv.js)

npm@5.6.0 (F:\Live\NODE\node\_global\node\_modules\cnpm\node\_modules\npm\lib\npm.js)

node@8.9.1 (F:\Live\NODE\node.exe)

npminstall@3.2.1 (F:\Live\NODE\node\_global\node\_modules\cnpm\node\_modules\npminstall\lib\index.js)

prefix=F:\Live\NODE\node\_global

win32 x64 10.0.16299

registry=http://registry.npm.taobao.org

但是我出现的是，cnpm不是外部命令。

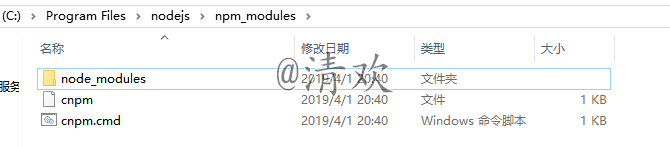
## 解决cnpm不是外部命令：

百度了一大堆。都是改目录或者改c盘下的npm的配置文件。

于是就想，出现这个结果是cnpm的路径没有被识别造成的。

于是cd到cnpm的路径中。（因为我之前安装npm的时候，配置过node\_modules和node\_cache。所以我就下意识的去nodejs的目录下去找。）

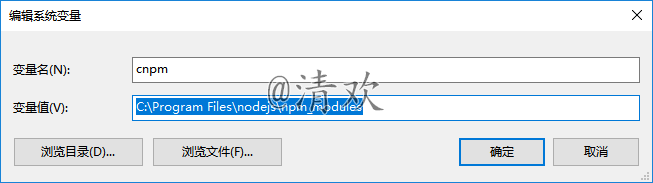
然后在这个路径下找到了：



就是安装nodejs路径下的，node\_global文件夹。

我们可以看到，这个cnpm和npm一样，有.cmd等等文件。cd到这个路径下运行cmd命令 cnpm -v之后，运行成功。

那么也就是说，之前直接输入cnpm -v 报的cnpm不是外部命令，**原因是系统找不到cnpm的可执行文件**。那也就是说自己给cnpm配个环境变量就好了。



右键电脑，属性，高级设置，环境变量，系统变量中的path里，最后加上这段话：

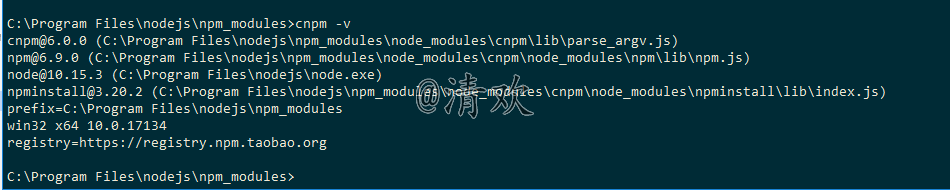
你的cnpm.cmd所在的路径再加上英文版的分号。*//这行文字其实是博客的代码格式，看，还有注释*

我的电脑中是这个：

C:\Program Files\nodejs\npm\_modules

就好了。

然后退出重新打开cmd。输入cnpm-v。搞定。注意，配置了环境变量之后要重新打开cmd啊。



## webpack失败

WARNING in configuration

The 'mode' option has not been set, webpack will fallback to 'production' for this value. Set 'mode' option to 'development' or 'production' to enable defaults for each environment.

You can also set it to 'none' to disable any default behavior. Learn more: https://webpack.js.org/concepts/mode/

ERROR in multi ./main.js build.js

Module not found: Error: Can't resolve 'build.js' in 'C:\java\xcEduUI01\xc-ui-pc-static-portal\webpacktest01'

@ multi ./main.js build.js main[1]

更改 命令为webpack main.js -o bulid.js

C:\java\xcEduUI01\xc-ui-pc-static-portal\webpacktest01>webpack main.js -o bulid.js

Hash: f59d4e004f8ed1268d9b

Version: webpack 4.29.6

Time: 310ms

Built at: 2019-04-02 08:20:30

Asset Size Chunks Chunk Names

bulid.js 1.16 KiB 0 [emitted] main

Entrypoint main = bulid.js

[0] ./main.js 635 bytes {0} [built]

WARNING in configuration

The 'mode' option has not been set, webpack will fallback to 'production' for this value. Set 'mode' option to 'development' or 'production' to enable defaults for each environment.

You can also set it to 'none' to disable any default behavior. Learn more: https://webpack.js.org/concepts/mode/

## Webpack4 打包出错问题解决

报错信息 :

WARNING in configuration

The ‘mode’ option has not been set, webpack will fallback to ‘production’ for this value. Set ‘mode’ option to ‘development’ or ‘production’ to enable defaults for each environment.

You can also set it to ‘none’ to disable any default behavior. Learn more: https://webpack.js.org/concepts/mode/

第一种常见解决方案: 因为在老版本中Webpack打包命令是

webpack .\src\main.js .\dist\bundle.js

而在webpack4之后 官方对 webpack-cli 单独抽离了出来 所以我们需要

1. 安装webpack

npm i webpack -g

2. 安装 cli

npm i -g webpack-cli

3. 打包 将 a.js b.js 替换了就可以

npx webpack a.js --output-filename b.js --output-path . --mode development

## 节点Sass尚不支持您当前的环境:Windows 64位

Cannot load gulp: Error: Node Sass does not yet support your current environment: Windows 64-bit with Unsupported runtime (57)

For more information on which environments are supported please see:

https://github.com/sass/node-sass/releases/tag/v3.13.1

[09:57:14] Cannot run sass task: missing in gulpfile.js

解决方法

cnpm install node-sass

成功界面

G:\Work\GateWay>cnpm install node-sass

√ Installed 1 packages

√ Linked 171 latest versions

Downloading binary from https://cdn.npm.taobao.org/dist/node-sass/v4.9.3/win32-x64-57\_binding.node

Download complete

Binary saved to G:\Work\GateWay\node\_modules\\_node-sass@4.9.3@node-sass\vendor\win32-x64-57\binding.node

Caching binary to C:\Users\Administrator\.npminstall\_tarball\node-sass\4.9.3\win32-x64-57\_binding.node

Binary found at G:\Work\GateWay\node\_modules\\_node-sass@4.9.3@node-sass\vendor\win32-x64-57\binding.node

Testing binary

Binary is fine

√ Run 1 scripts

√ All packages installed (184 packages installed from npm registry, used 13s, speed 307.17kB/s, json 172(1.35MB), tarball 2.68MB)