# 前端开发和前端开发工具

## 一、前端开发

前端工程师“Front-End-Developer”源自于美国。大约从2005年开始正式的前端工程师角色被行业所认可，到了2010年，互联网开始全面进入移动时代，前端开发的工作越来越重要。

最初所有的开发工作都是由后端工程师完成的，随着业务越来越繁杂，工作量变大，于是我们将项目中的可视化部分和一部分交互功能的开发工作剥离出来，形成了前端开发。

由于互联网行业的急速发展，导致了在不同的国家，有着截然不同的分工体制。

在日本和一些人口比较稀疏的国家，例如加拿大、澳洲等，流行“Full-Stack Engineer”，也就是我们通常所说的全栈工程师。通俗点说就是一个人除了完成前端开发和后端开发工作以外，有的公司从产品设计到项目开发再到后期运维可能都是同一个人，甚至可能还要负责UI、配动画，也可以是扫地、擦窗、写文档、维修桌椅等等。

而在美国等互联网环境比较发达的国家项目开发的分工协作更为明确，整个项目开发分为前端、中间层和后端三个开发阶段，这三个阶段分别由三个或者更多的人来协同完成。

国内的大部分互联网公司只有前端工程师和后端工程师，中间层的工作有的由前端来完成，有的由后端来完成。

PRD（产品原型-产品经理） - PSD（视觉设计-UI工程师） - HTML/CSS/JavaScript（PC/移动端网页，实现网页端的视觉展示和交互-前端工程师）

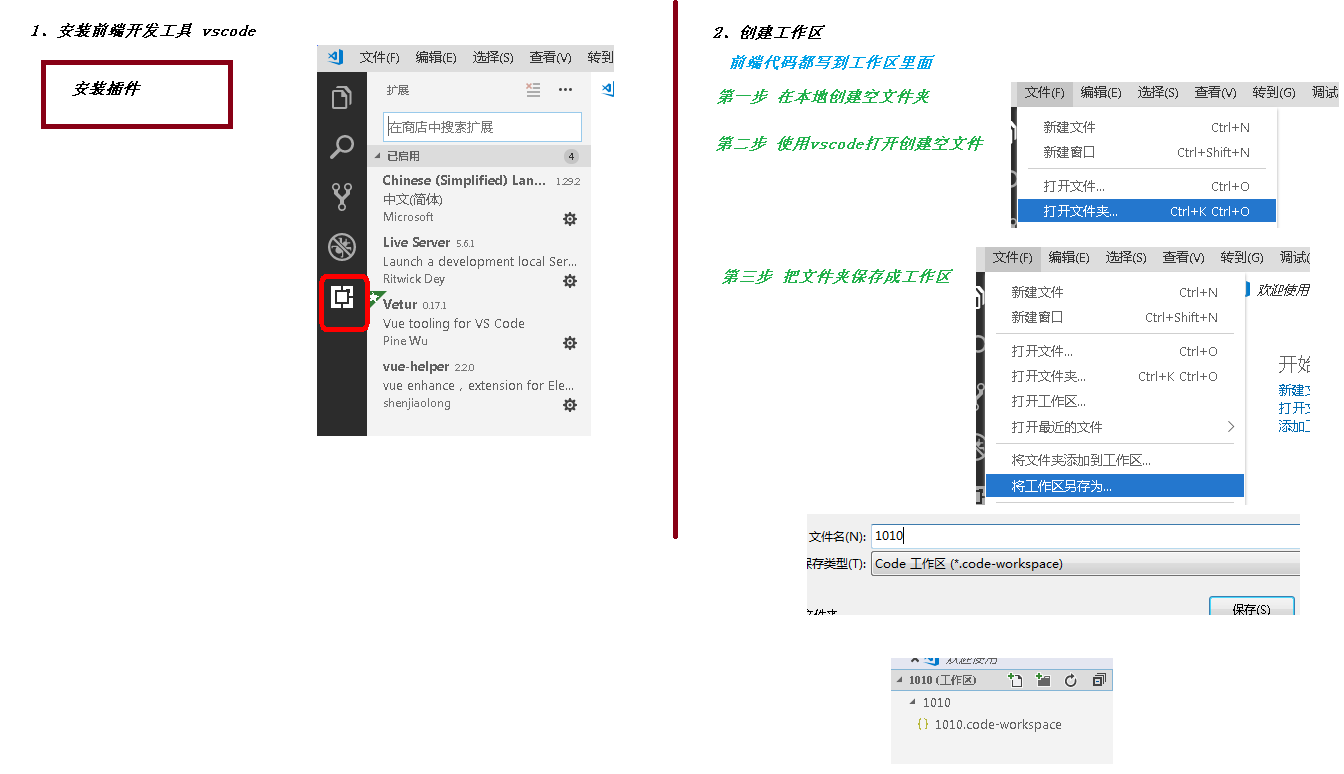
## 二、下载和安装VS Code

### 1、下载地址

<https://code.visualstudio.com/>

### 2、安装

## 三、初始设置



### 1、中文界面配置

* 首先安装中文插件：Chinese (Simplified) Language Pack for Visual Studio Code
* 右下角弹出是否重启vs，点击“yes”
* 有些机器重启后如果界面没有变化，则 点击 左边栏Manage -> Command Paletet...【Ctrl+Shift+p】
* 在搜索框中输入“configure display language”，回车
* 打开locale.json文件，修改文件下的属性 "locale":"zh-cn"

{

// 定义 VS Code 的显示语言。

// 请参阅 https://go.microsoft.com/fwlink/?LinkId=761051，了解支持的语言列表。

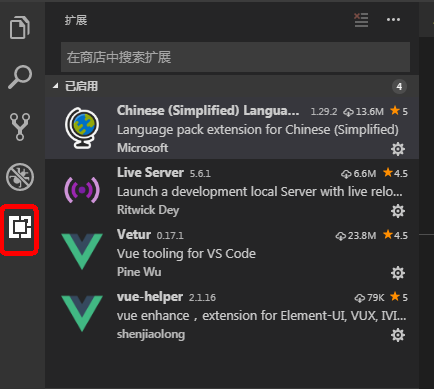
"locale":"zh-cn" // 更改将在重新启动 VS Code 之后生效。

}

* 重启vs

### 2、插件安装

为方便后续开发，建议安装如下插件（红色矩形框标记的插件）



**3、创建项目**

vscode本身没有新建项目的选项，所以要先创建一个空的文件夹，如project\_xxxx。

然后打开vscode，再在vscode里面选择 File -> Open Folder 打开文件夹，这样才可以创建项目。

### 4、保存工作区

打开文件夹后，选择“文件 -> 将工作区另存为...”，为工作区文件起一个名字，存储在刚才的文件夹下即可

### 5、新建文件夹和网页

C:\Users\gouwenyong0000\Documents\My Knowledge\temp\8d7d110a-4c04-4911-a508-a7205844fe0f\128\index_files\b5631e4b-0ce6-4ed7-ba31-c9b88f8aa53d.png

### 6、预览网页

**以文件路径方式打开网页预览**

需要安装“open in browser”插件：

文件右键 -> Open In Default Browser

**以服务器方式打开网页预览**

需要安装“Live Server”插件：

文件右键 -> Open with Live Server

### 7、设置字体大小

左边栏Manage -> settings -> 搜索 “font” -> Font size

### 8、开启完整的Emmet语法支持

设置中搜索 Emmet：启用如下选项，必要时重启vs



# ES6入门

自学参考：<http://es6.ruanyifeng.com/>

## 一、ECMAScript 6 简介

ECMAScript 6.0（以下简称 ES6）是 JavaScript 语言的下一代标准，已经在 2015 年 6 月正式发布了。它的目标，是使得 JavaScript 语言可以用来编写复杂的大型应用程序，成为企业级开发语言。

### 1、ECMAScript 和 JavaScript 的关系

一个常见的问题是，ECMAScript 和 JavaScript 到底是什么关系？

要讲清楚这个问题，需要回顾历史。1996 年 11 月，JavaScript 的创造者 Netscape 公司，决定将 JavaScript 提交给标准化组织 ECMA，希望这种语言能够成为国际标准。次年，ECMA 发布 262 号标准文件（ECMA-262）的第一版，规定了浏览器脚本语言的标准，并将这种语言称为 ECMAScript，这个版本就是 1.0 版。

因此，ECMAScript 和 JavaScript 的关系是，前者是后者的规格，后者是前者的一种实现（另外的 ECMAScript 方言还有 Jscript 和 ActionScript）

### 2、ES6 与 ECMAScript 2015 的关系

ECMAScript 2015（简称 ES2015）这个词，也是经常可以看到的。它与 ES6 是什么关系呢？

2011 年，ECMAScript 5.1 版发布后，就开始制定 6.0 版了。因此，ES6 这个词的原意，就是指 JavaScript 语言的下一个版本。

ES6 的第一个版本，在 2015 年 6 月发布，正式名称是《ECMAScript 2015 标准》（简称 ES2015）。

2016 年 6 月，小幅修订的《ECMAScript 2016 标准》（简称 ES2016）如期发布，这个版本可以看作是 ES6.1 版，因为两者的差异非常小，基本上是同一个标准。根据计划，2017 年 6 月发布 ES2017 标准。

因此，ES6 既是一个历史名词，也是一个泛指，含义是 5.1 版以后的 JavaScript 的下一代标准，涵盖了 ES2015、ES2016、ES2017 等等，而 ES2015 则是正式名称，特指该年发布的正式版本的语言标准。本书中提到 ES6 的地方，一般是指 ES2015 标准，但有时也是泛指“下一代 JavaScript 语言”。

## 二、基本语法

ES标准中不包含 DOM 和 BOM的定义，只涵盖基本数据类型、关键字、语句、运算符、内建对象、内建函数等通用语法。

本部分只学习前端开发中ES6的最少必要知识，方便后面项目开发中对代码的理解。

### 1、let声明变量

创建 let.html

<script>

    // var 声明的变量没有局部作用域

    // let 声明的变量  有局部作用域

    {

*var* a = 0

*let* b = 1

    }

    console.log(a)  // 0

    console.log(b)  // ReferenceError: b is not defined

    // var 可以声明多次

    // let 只能声明一次

*var* m = 1

*var* m = 2

*let* n = 3

*let* n = 4

    console.log(m)  // 2

    console.log(n)  // Identifier 'n' has already been declared

</script>

### 2、const声明常量（只读变量）

创建 const.html

<script>

    // 1、声明之后不允许改变

*const* PI = "3.1415926"

    PI = 3  // TypeError: Assignment to constant variable.

    // 2、一但声明必须初始化，否则会报错

*const* MY\_AGE  // SyntaxError: Missing initializer in const declaration

</script>

### 3、解构赋值

创建 解构赋值.html

解构赋值是对赋值运算符的扩展。

他是一种针对数组或者对象进行模式匹配，然后对其中的变量进行赋值。

在代码书写上简洁且易读，语义更加清晰明了；也方便了复杂对象中数据字段获取。

<script>

    //1、数组解构

    // 传统

*let* a = 1, b = 2, c = 3

    console.log(a, b, c)

    // ES6

*let* [x, y, z] = [1, 2, 3]

    console.log(x, y, z)

    //2、对象解构

*let* user = { name: 'Helen', age: 18 }

    // 传统

*let* name1 = user.name

*let* age1 = user.age

    console.log(name1, age1)

    // ES6

*let* { name, age } = user//注意：结构的变量必须是user中的属性

    console.log(name, age)

</script>

### 4、模板字符串

创建 模板字符串.html

模板字符串相当于加强版的字符串，用反引号 `,除了作为普通字符串，还可以用来定义多行字符串，还可以在字符串中加入变量和表达式。

    // 1、多行字符串

*let* string1 = `Hey,

can you stop angry now?`

    console.log(string1)

    // Hey,

    // can you stop angry now?

    // 2、字符串插入变量和表达式。变量名写在 ${} 中，${} 中可以放入 JavaScript 表达式。

*let* name = "Mike"

*let* age = 27

*let* info = `My Name is ${name},I am ${age + 1} years old next year.`

    console.log(info)

    // My Name is Mike,I am 28 years old next year.

    // 3、字符串中调用函数

*function* f() {

        return "have fun!"

    }

*let* string2 = `Game start,${f()}`

    console.log(string2);  // Game start,have fun!

### 5、声明对象简写

创建 声明对象简写.html

<script>

*const* age = 12

*const* name = "Amy"

    // 传统

*const* person1 = { age: age, name: name }

    console.log(person1)

    // ES6

*const* person2 = { age, name }

    console.log(person2) //{age: 12, name: "Amy"}

</script>

### 6、定义方法简写

创建 定义方法简写.html

<script>

    // 传统

*const* person1 = {

        sayHi: *function* () {

            console.log("Hi")

        }

    }

    person1.sayHi();//"Hi"

    // ES6

*const* person2 = {

        sayHi() {

            console.log("Hi")

        }

    }

    person2.sayHi()  //"Hi"

</script>

### 7、对象拓展运算符

创建 对象拓展运算符.html

拓展运算符（...）用于取出参数对象所有可遍历属性然后拷贝到当前对象。

 <script>

    // 1、拷贝对象

*let* person1 = { name: "Amy", age: 15 }

*let* someone = { ...person1 }

    console.log(someone)  //{name: "Amy", age: 15}

    // 2、合并对象

*let* age = { age: 15 }

*let* name = { name: "Amy" }

*let* person2 = { ...age, ...name }

    console.log(person2)  //{age: 15, name: "Amy"}

</script>

### 8、箭头函数

创建 箭头函数.html

箭头函数提供了一种更加简洁的函数书写方式。基本语法是：

参数 => 函数体

<script>

    // 传统

*var* f1 = *function* (*a*) {

        return *a*

    }

    console.log(f1(1))

    // ES6

*var* f2 = *a* *=>* *a*

    console.log(f2(1))

    // 当箭头函数没有参数或者有多个参数，要用 () 括起来。

    // 当箭头函数函数体有多行语句，用 {} 包裹起来，表示代码块，

    // 当只有一行语句，并且需要返回结果时，可以省略 {} , 结果会自动返回。

*var* f3 = (*a*, *b*) *=>* {

*let* result = *a* + *b*

        return result

    }

    console.log(f3(6, 2))  // 8

    // 前面代码相当于：

*var* f4 = (*a*, *b*) *=>* *a* + *b*

</script>

箭头函数多用于匿名函数的定义

### 拓展：定时器案例

<body>

    <h1 id="show">100</h1>

    <script>

*var* n = 100;

        setInterval(() *=>* {

            document.getElementById("show").innerText = n--;

        }, 1000);

    </script>

</body>

</html>

      //倒计时方法 vue中使用

      timeDown() {

*let* result = setInterval(() *=>* {

          --this.second;

          this.codeTest = this.second

          if (this.second < 1) { //终止 恢复初始数据

            clearInterval(result);

            this.sending = true;

            //this.disabled = false;

            this.second = 60;

            this.codeTest = "获取验证码"

          }

        }, 1000);

      },

# Vue

## 一、介绍

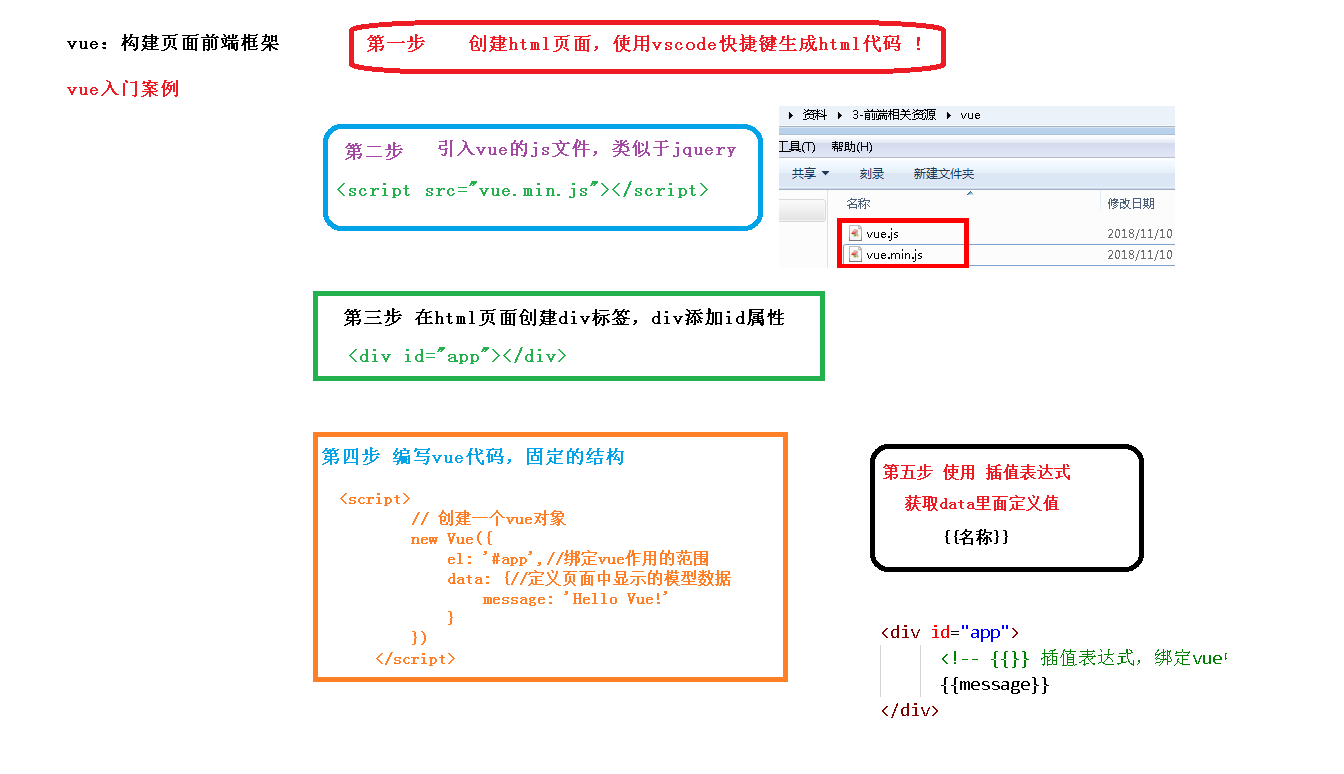
### 1、Vue.js 是什么

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。

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。

官方网站：<https://cn.vuejs.org>

### 2、初始Vue.js



创建 demo.html

<!-- id标识vue作用的范围 -->

<div id="app">

    <!-- {{}} 插值表达式，绑定vue中的data数据  v-text指令 -->

    {{ message }}

</div>

<script src="vue.min.js"></script>

<script>

    // 创建一个vue对象

    new Vue({

        el: '#app',//绑定vue作用的范围

        data: {//定义页面中显示的模型数据

            message: 'Hello Vue!'

        }

    })

</script>

这就是声明式渲染：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

这里的核心思想就是没有繁琐的DOM操作，例如jQuery中，我们需要先找到div节点，获取到DOM对象，然后进行一系列的节点操作

**在vs code中创建代码片段：**

文件 =>  首选项 => 用户代码片段 => 新建全局代码片段/或文件夹代码片段：vue-html.code-snippets

**注意：制作代码片段的时候，字符串中如果包含文件中复制过来的“Tab”键的空格，要换成“空格键”的空格**

{

*"vue htm"*: {

*"scope"*: "html",

*"prefix"*: "vuehtml",

*"body"*: [

            "<!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>",

            "</head>",

            "",

            "<body>",

            "    <div id=\"app\">",

            "",

            "    </div>",

            "    <script src=\"vue.min.js\"></script>",

            "    <script>",

            "        new Vue({",

            "            el: '#app',",

            "            data: {",

            "                $1",

            "            }",

            "        })",

            "    </script>",

            "</body>",

            "",

            "</html>",

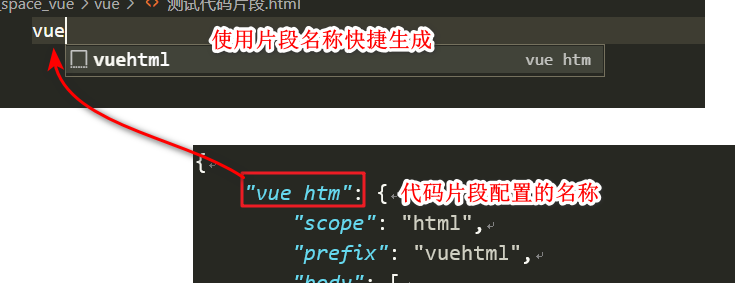
        ],

*"description"*: "my vue template in html"

    }

}

使用代码片段：配置的快捷提示



## 二、基本语法

### 1、基本数据渲染和指令

创建 01-基本数据渲染和指令.html

你看到的 v-bind 特性被称为指令。指令带有前缀 v-

除了使用插值表达式{{}}进行数据渲染，也可以使用 v-bind指令， 单向数据绑定

它的简写的形式就是一个冒号（:）

            data: {

                content: '我是标题',

                message: '页面加载于 ' + new Date().toLocaleString()

            }

 <!-- 如果要将模型数据绑定在html属性中，则使用 v-bind 指令  单向数据绑定

此时title中显示的是模型数据

-->

<h1 v-bind:title="message">

    {{content}}

</h1>

<!-- v-bind 指令的简写形式： 冒号（:） -->

<h1 :title="message">

    {{content}}

</h1>

### 2、双向数据绑定

创建 02-双向数据绑定.html

双向数据绑定和单向数据绑定：使用 v-model 进行双向数据绑定

   <div id="app">

        <!-- v-bind:value只能进行单向的数据渲染  一般用于属性-->

        <input type="text" v-bind:value="searchMap.keyWord">

        <!-- v-model 可以进行双向的数据绑定  一般用户表单-->

        <input type="text" **v-model="searchMap.keyWord"**>

        <p>您要查询的是：{{searchMap.keyWord}}</p>

    </div>

    <script src="vue.min.js"></script>

    <script>

        new Vue({

            el: '#app',

            data: {

                searchMap: {

                    keyWord: '尚硅谷'

                }

            }

        })

    </script>

### 3、事件

创建 03-事件.html

**需求：**点击查询按钮，按照输入框中输入的内容查找公司相关信息

在前面的例子基础上，data节点中增加 result，增加 methods节点 并定义 search方法

data: {

searchMap:{

keyWord: '尚硅谷'

},

//查询结果

result: {}

},

methods:{

search(){

console.log('search')

//TODO

}

}

html中增加 button 和 p

使用 v-on 进行数件处理，v-on:click 表示处理鼠标点击事件，事件调用的方法定义在 vue 对象声明的 methods 节点中

<!-- v-on 指令绑定事件，click指定绑定的事件类型，事件发生时调用vue中methods节点中定义的方法  括号可以不加-->

<button v-on:click="search()">查询</button>

<p>您要查询的是：{{searchMap.keyWord}}</p>

<p><a v-bind:href="result.site" target="\_blank" v-text="result.title"></a></p>

完善search方法

methods: {

    serach() {

        console.log("serach.........")

        this.result = {

            "title": "尚硅谷",

            "site": "http://www.atguigu.com"

        }

    }

}

简写

<!-- v-on 指令的简写形式 @ -->

<button @click="search">查询</button>

### 4、修饰符

创建 04-修饰符.html

修饰符 (Modifiers) 是以半角句号（.）指明的特殊后缀，用于指出一个指令应该以特殊方式绑定。

例如，.prevent 修饰符告诉 v-on 指令对于触发的事件调用 event.preventDefault()：

即**阻止事件原本的默认行为**

   <div id="app">

        <!-- 修饰符用于指出一个指令应该以特殊方式绑定。

     这里的 .prevent 修饰符告诉 v-on 指令对于触发的事件调用js的 event.preventDefault()：

     即阻止表单提交的默认行为 -->

        <form action="save" v-on:submit.prevent="onSubmit" method="GET">

            <label for="username">

                <input type="text" id="username" v-model="user.username">

                <button type="submit">保存</button>

            </label>

        </form>

    </div>

    <script src="vue.min.js"></script>

    <script>

        new Vue({

            el: '#app',

            data: {

                user: {}

            },

            methods: {

                onSubmit() {

                    if (this.user.username) {

                        console.log('提交表单')

                    } else {

                        alert('请输入用户名')

                    }

                }

            }

        })

    </script>

### 5、条件渲染

创建 05-条件渲染.html

v-if：条件指令 控制dom的显示

data: {

ok: false

}

注意：单个复选框绑定到布尔值

        <input type="checkbox" v-model="ok">同意许可协议

        <!-- v:if条件指令：还有v-else、v-else-if 切换开销大 -->

        <h1 v-if="ok == true">if：Lorem ipsum dolor sit amet.</h1>

        <h1 v-else>no</h1>

v-show：条件指令

使用v-show完成和上面相同的功能

        <!-- v:show 条件指令 初始渲染开销大 -->

        <h1 v-show="ok">show：Lorem ipsum dolor sit amet.</h1>

        <h1 v-show="ok == true">no</h1>

* v-if 是“真正”的条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建。
* v-if 也是**惰性的**：如果在初始渲染时条件为假，则什么也不做——直到条件第一次变为真时，才会开始渲染条件块。
* 相比之下，v-show 就简单得多——不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 进行切换。
* 一般来说，v-if 有更高的切换开销，而 v-show 有更高的初始渲染开销。因此，如果需要非常频繁地切换，则使用 v-show 较好；如果在运行时条件很少改变，则使用 v-if 较好。

### 6、列表渲染

创建 06-列表渲染.html

v-for：列表循环指令

**例1：简单的列表渲染**

<!-- 1、简单的列表渲染 -->

<ul>

    <li v-for=" item in 10">{{ item }} </li>

</ul>

<ul>

    <!-- 如果想获取索引，则使用index关键字，注意，圆括号中的index必须放在后面 -->

    <li v-for="(item,index) in 5">{{ item }} - {{ index }} </li>

</ul>

**例2：遍历数据列表**

data: {

userList: [

{ id: 1, username: 'helen', age: 18 },

{ id: 2, username: 'peter', age: 28 },

{ id: 3, username: 'andy', age: 38 }

]

}

<table border="1">

    <!-- <tr v-for="item in userList"></tr> -->

    <tr v-for="(item, index) in userList">

        <td>{{index}}</td>

        <td>{{item.id}}</td>

        <td>{{item.username}}</td>

        <td>{{item.age}}</td>

    </tr>

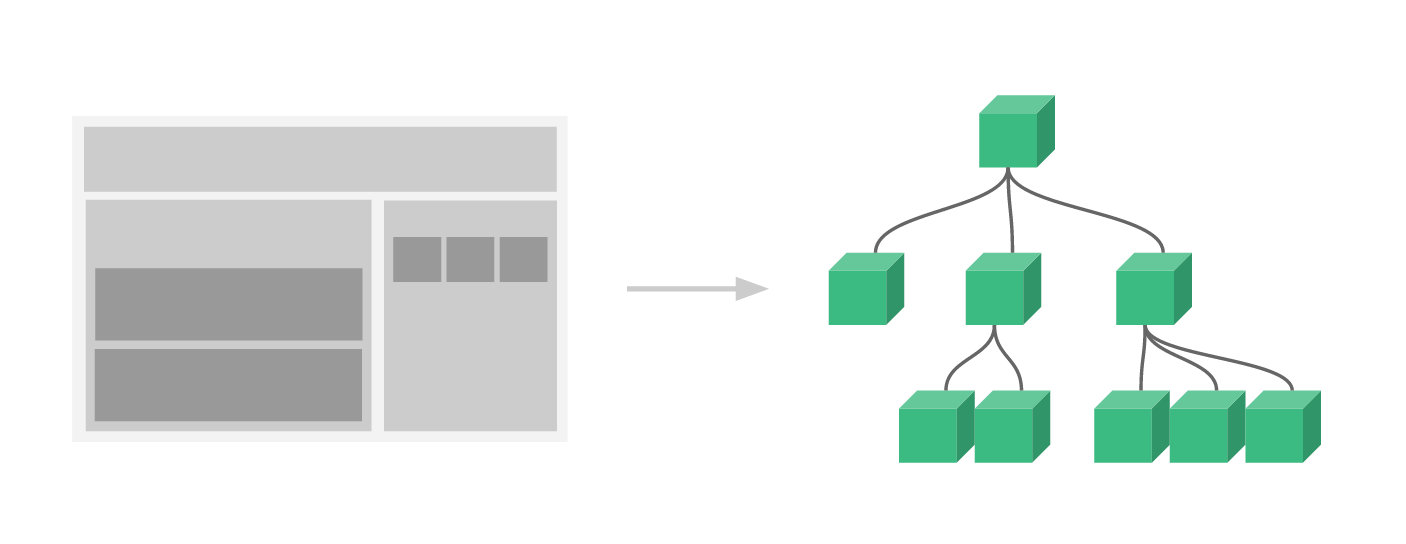
</table>

## 三、组件（重点）

组件（Component）是 Vue.js 最强大的功能之一。

组件可以扩展 HTML 元素，封装可重用的代码。

组件系统让我们可以用独立可复用的小组件来构建大型应用，几乎任意类型的应用的界面都可以抽象为一个组件树：



### 1、局部组件

创建 01-1-局部组件.html

定义组件

*var* app = new Vue({

    el: '#app',

    // 定义局部组件，这里可以定义多个局部组件

    components: {

        //组件的名字

        'Navbar': {

            //组件的内容

            template: '<ul><li>首页</li><li>学员管理</li></ul>'

        }

    }

})

使用组件

    <div id="app">

        <Navbar></Navbar>

    </div>

### 2、全局组件

创建 01-2-全局组件.html

定义全局组件：components/Navbar.js

// 定义全局组件

Vue.component('Navbar', {

    template: '<ul><li>首页</li><li>学员管理</li><li>讲师管理</li></ul>'

})

 引入全局组件的js文件即可使用

<div id="app">

    <Navbar></Navbar>

</div>

<script src="vue.min.js"></script>

<script src="components/Navbar.js"></script><!--引入相应js文件--->

<script>

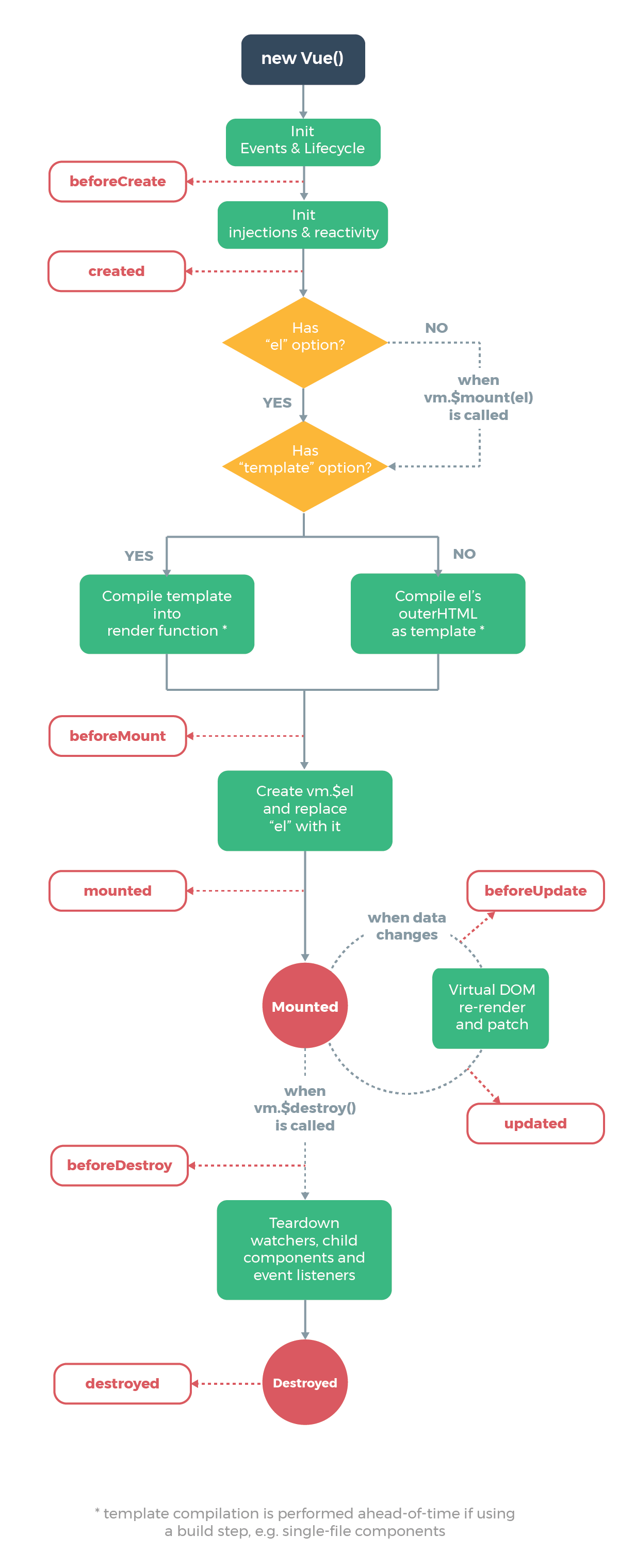
*var* app = new Vue({

        el: '#app'

    })

</script>

## 四、实例生命周期

  
    <div id="app">

        <p id="pp">{{msg}}</p>

    </div>

    <script src="vue.min.js"></script>

    <script>

        new Vue({

            el: '#app',

            data: {

                msg: "hello"

            },

            created() {

                //页面渲染之前执行

                console.log("cccccccc............")

                console.log(document.getElementById("pp").innerText) //控制台打印：{{msg}}

            },

            mounted() {

                //页面渲染之后执行

                console.log("mmmmmm............")

                console.log(document.getElementById("pp").innerText) //控制台打印：hello

            }

        })

    </script>

创建 03-vue实例的生命周期.html

 data: {

     message: '床前明月光'

 },

 methods: {

     show() {

         console.log('执行show方法')

     },

     update() {

         this.message = '玻璃好上霜'

     }

 },

<button @click="update">update</button>

<h3 id="h3">{{ message }}</h3>

分析生命周期相关方法的执行时机

//===创建时的四个事件

beforeCreate() { // 第一个被执行的钩子方法：实例被创建出来之前执行

    console.log(this.message) //undefined

    this.show() //TypeError: this.show is not a function

    // beforeCreate执行时，data 和 methods 中的 数据都还没有没初始化

},

created() { // 第二个被执行的钩子方法 页面渲染之前执行

    console.log(this.message) //床前明月光

    this.show() //执行show方法

    // created执行时，data 和 methods 都已经被初始化好了！

    // 如果要调用 methods 中的方法，或者操作 data 中的数据，最早，只能在 created 中操作

},

beforeMount() { // 第三个被执行的钩子方法

    console.log(document.getElementById('h3').innerText) //{{ message }}

    // beforeMount执行时，模板已经在内存中编辑完成了，尚未被渲染到页面中

},

mounted() { // 第四个被执行的钩子方法 页面渲染之后执行

    console.log(document.getElementById('h3').innerText) //床前明月光

    // 内存中的模板已经渲染到页面，用户已经可以看见内容

},

//===运行中的两个事件

beforeUpdate() { // 数据更新的前一刻

    console.log('界面显示的内容：' + document.getElementById('h3').innerText)

    console.log('data 中的 message 数据是：' + this.message)

    // beforeUpdate执行时，内存中的数据已更新，但是页面尚未被渲染

},

updated() {

    console.log('界面显示的内容：' + document.getElementById('h3').innerText)

    console.log('data 中的 message 数据是：' + this.message)

    // updated执行时，内存中的数据已更新，并且页面已经被渲染

}

## 五、路由

Vue.js 路由允许我们通过不同的 URL 访问不同的内容。

通过 Vue.js 可以实现多视图的单页Web应用（single page web application，SPA）。

Vue.js 路由需要载入 vue-router 库

创建 04-路由.html

### 1、引入js

    <script src="vue.min.js"></script>

    <script src="./vue-router.min.js"></script><!-- 先后顺序 -->

### 2、编写html

<div id="app">

    <h1>Hello App!</h1>

    <p>

        <!-- 使用 router-link 组件来导航. -->

        <!-- 通过传入 `to` 属性指定链接. -->

        <!-- <router-link> 默认会被渲染成一个 `<a>` 标签 -->

        <router-link to="/">首页</router-link>

        <router-link to="/student">会员管理</router-link>

        <router-link to="/teacher">讲师管理</router-link>

    </p>

    <!-- 路由出口 -->

    <!-- 路由匹配到的组件将渲染在这里 -->

    <router-view></router-view>

</div>

### 3、编写js 路由初始化

    <script>

        // 1. 定义（路由）组件。

        // 可以从其他文件 import 进来

*const* Welcome = { template: '<div>欢迎</div>' }

*const* Student = { template: '<div>student list</div>' }

*const* Teacher = { template: '<div>teacher list</div>' }

        // 2. 定义路由

        // 每个路由应该映射一个组件。

*const* routes = [

            { path: '/', redirect: '/welcome' }, //设置默认指向的路径

            { path: '/welcome', component: Welcome },

            { path: '/student', component: Student },

            { path: '/teacher', component: Teacher }

        ]

        // 3. 创建 router 实例，然后传 `routes` 配置

*const* router = new VueRouter({

            routes // （缩写）相当于 routes: routes

        })

        // 4. 创建和挂载根实例。

        // 从而让整个应用都有路由功能

*const* app = new Vue({

            el: '#app',

            router

        })

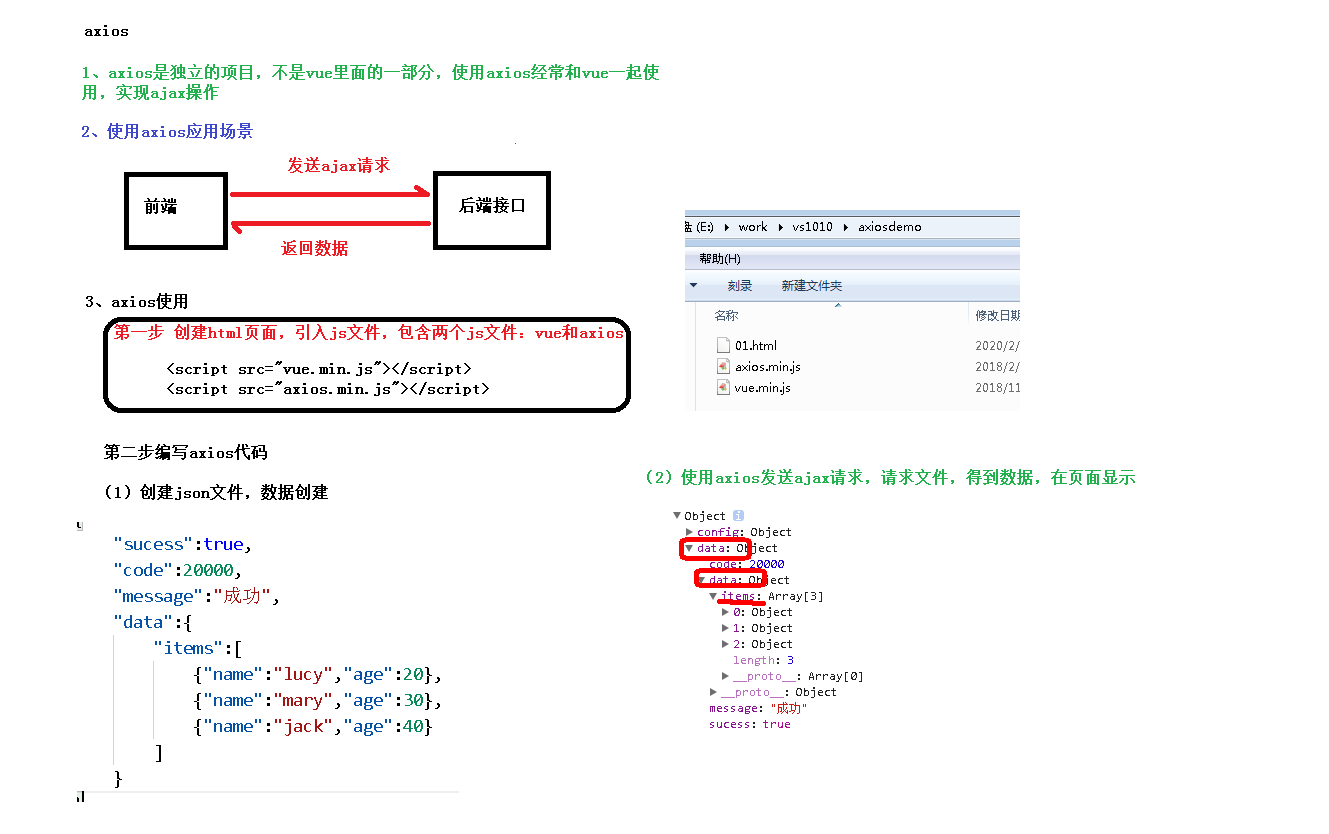
        // 现在，应用已经启动了！

    </script>

## 六、axios

axios是独立于vue的一个项目，基于promise用于浏览器和node.js的http客户端

* 在浏览器中可以帮助我们完成 ajax请求的发送
* 在node.js中可以向远程接口发送请求



### 1、获取数据

    <script src="vue.min.js"></script>

    <script src="axios.min.js"></script>

注意：测试时需要开启后端服务器，并且后端开启跨域访问权限

 模拟数据：data.json

{

*"sucess"*:true,

*"code"*:20000,

*"message"*:"成功",

*"data"*:{

*"items"*:[

            {*"name"*:"lucy",*"age"*:20},

            {*"name"*:"mary",*"age"*:30},

            {*"name"*:"jack",*"age"*:40}

        ]

    }

}

### 2、axios.get()请求数据

new Vue({

    el: '#app',

    data: { //在data定义变量和初始化

        memberList: [] //数组

    },

    created() { //页面渲染之前执行

        //调用定义的方法

        this.getList()

    },

    methods: { //具体方法

        //创建方法  查询所有用户数据

        getList() {

            //使用axios发送ajax请求

            //axios.提交方式("请求路径").then().catch()

            axios.get("./data.json")

                .then((*response*) *=>* {//请求成功的then

                    console.log(*response*)

                    this.memberList = *response*.data.data.items;

                })

                .catch((*err*) *=>* {//请求失败的执行catch

                    alert("faild")

                });

        }

    }

})

**控制台查看输出**

### 3、页面展示数据

<div id="app">

    <table border="1">

        <tr>

            <td>序号</td>

            <td>姓名</td>

            <td>年龄</td>

        </tr>

        <tr v-for="(item,index) in memberList">

            <td>{{index}}</td>

            <td>{{item.name}}</td>

            <td>{{item.age}}</td>

        </tr>

    </table>

</div>

## 七、element-ui：

element-ui 是饿了么前端出品的基于 Vue.js的 后台组件库，方便程序员进行页面快速布局和构建

官网： <http://element-cn.eleme.io/#/zh-CN>

创建 06-element-ui.html

将element-ui引入到项目

C:\Users\gouwenyong0000\Documents\My Knowledge\temp\385c7951-b5d5-43fb-8334-f7ddc87ea984\128\index_files\67c16425-795a-48bb-ad7d-e0b8fa1c8ea5.png

### 1、引入css

<!-- import CSS -->

<link rel="stylesheet" href="element-ui/lib/theme-chalk/index.css">

### 2、引入js

<!-- import Vue before Element -->

<script src="vue.min.js"></script>

<!-- import JavaScript -->

<script src="element-ui/lib/index.js"></script>

### 3、编写html

<div id="app">

<el-button @click="visible = true">Button</el-button>

<el-dialog :visible.sync="visible" title="Hello world">

<p>Try Element</p>

</el-dialog>

</div>

关于.sync的扩展阅读

<https://www.jianshu.com/p/d42c508ea9de>

### 4、编写js

<script>

new Vue({

el: '#app',

data: function () {//定义Vue中data的另一种方式

return { visible: false }

}

})

</script>

**测试**

其他ui组件我们在项目中学习

# Node



## 一、简介

### 1、什么是Node.js

简单的说 Node.js 就是运行在服务端的 JavaScript。

Node.js是一个事件驱动I/O服务端JavaScript环境，基于Google的V8引擎，V8引擎执行Javascript的速度非常快，性能非常好。

### 2、Node.js有什么用

如果你是一个前端程序员，你不懂得像PHP、Python或Ruby等动态编程语言，然后你想创建自己的服务，那么Node.js是一个非常好的选择。

Node.js 是运行在服务端的 JavaScript，如果你熟悉Javascript，那么你将会很容易的学会Node.js。

当然，如果你是后端程序员，想部署一些高性能的服务，那么学习Node.js也是一个非常好的选择。

## 二、安装

### 1、下载

官网：<https://nodejs.org/en/>

中文网：<http://nodejs.cn/>

LTS：长期支持版本

Current：最新版

### 2、安装

### 3、查看版本

node -v

## 三、快速入门

### 1、创建文件夹nodejs

**2、控制台程序**

创建 01-控制台程序.js

console.log('Hello Node.js')

打开命令行终端：Ctrl + Shift + y

进入到程序所在的目录，输入

*node 01-控制台程序.js*

浏览器的内核包括两部分核心：

* DOM渲染引擎；
* js解析器（js引擎）
* js运行在浏览器中的内核中的js引擎内部

Node.js是脱离浏览器环境运行的JavaScript程序，基于V8 引擎（Chrome 的 JavaScript的引擎）

### 3、服务器端应用开发（了解）

创建 02-server-app.js

*const* http = require('http');

http.createServer(*function* (*request*, *response*) {

    // 发送 HTTP 头部

    // HTTP 状态值: 200 : OK

    // 内容类型: text/plain

*response*.writeHead(200, { 'Content-Type': 'text/plain' });

    // 发送响应数据 "Hello World"

*response*.end('Hello Server');

}).listen(8888);

// 终端打印如下信息

console.log('Server running at http://127.0.0.1:8888/');

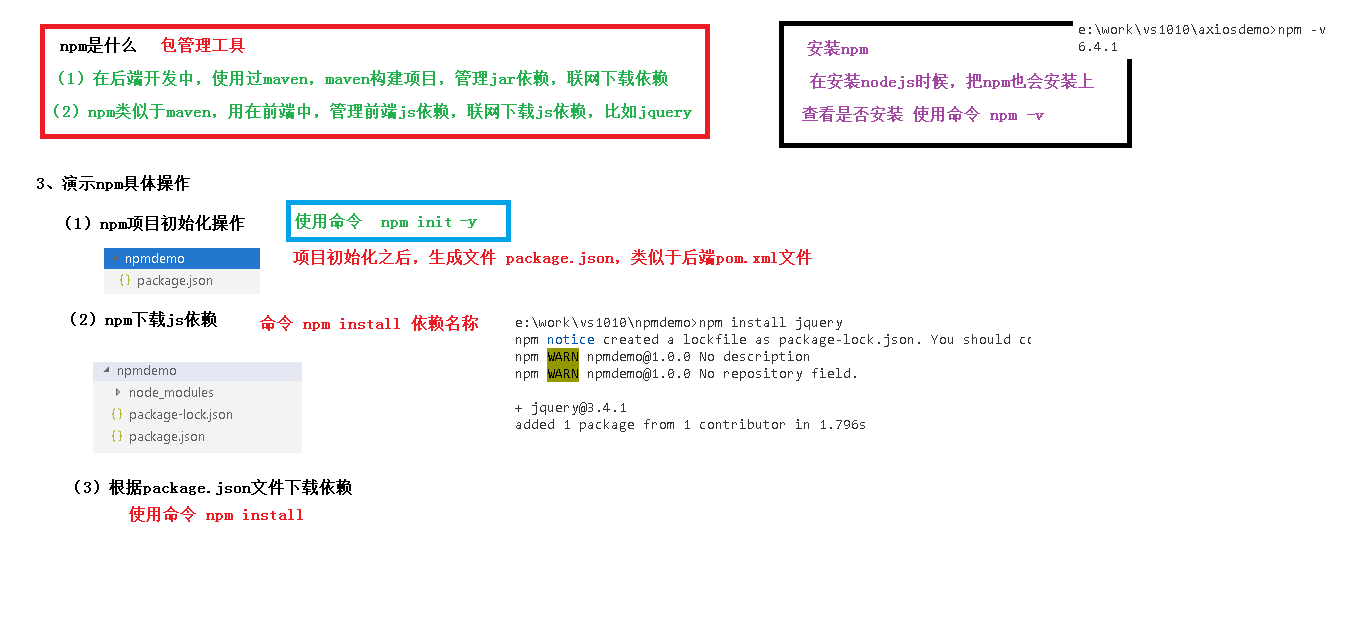
运行服务器程序

*node 02-server-app.js*

服务器启动成功后，在浏览器中输入：<http://localhost:8888/> 查看webserver成功运行，并输出html页面

停止服务：ctrl + c

# NPM包管理器



## 一、简介

### 1、什么是NPM

NPM全称Node Package Manager，是Node.js包管理工具，是全球最大的模块生态系统，里面所有的模块都是开源免费的；也是Node.js的包管理工具，相当于前端的Maven 。

### 2、NPM工具的安装位置

我们通过npm 可以很方便地下载js库，管理前端工程。

Node.js默认安装的npm包和工具的位置：Node.js目录\node\_modules

* 在这个目录下你可以看见 npm目录，npm本身就是被NPM包管理器管理的一个工具，说明 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*

*npm -v*

## 二、使用npm管理项目

### 1、创建文件夹npm

### 2、项目初始化npm init

#建立一个空文件夹，在命令提示符进入该文件夹  执行命令初始化

npm init

#按照提示输入相关信息，如果是用默认值则直接回车即可。

#name: 项目名称

#version: 项目版本号

#description: 项目描述

#keywords: {Array}关键词，便于用户搜索到我们的项目

#最后会生成package.json文件，这个是包的配置文件，相当于maven的pom.xml

#我们之后也可以根据需要进行修改。

#如果想直接生成默认 package.json 文件，那么可以使用命令

npm init -y

### 2、修改npm镜像

NPM官方的管理的包都是从 <http://npmjs.com>下载的，但是这个网站在国内速度很慢。

这里推荐使用淘宝 NPM 镜像 <http://npm.taobao.org/> ，淘宝 NPM 镜像是一个完整 npmjs.com 镜像，同步频率目前为 10分钟一次，以保证尽量与官方服务同步。

**设置镜像地址：**

#经过下面的配置，以后所有的 npm install 都会经过淘宝的镜像地址下载

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

#查看npm配置信息

npm config list

# 删除设置

npm config delete registry

### 3、npm install命令的使用

#使用 npm install 安装依赖包的最新版，

#模块安装的位置：项目目录\node\_modules

#安装会自动在项目目录下添加 package-lock.json文件，这个文件帮助锁定安装包的版本

#同时package.json 文件中，依赖包会被添加到dependencies节点下，类似maven中的 <dependencies>

npm install jquery

#如果安装时想指定特定的版本

npm install jquery@2.1.x

#npm管理的项目在备份和传输的时候一般不携带node\_modules文件夹

npm install #根据package.json中的配置下载依赖，初始化项目

#devDependencies节点：开发时的依赖包，项目打包到生产环境的时候不包含的依赖

#使用 -D参数将依赖添加到devDependencies节点

npm install --save-dev eslint

#或

npm install -D eslint

#全局安装

#Node.js全局安装的npm包和工具的位置：用户目录\AppData\Roaming\npm\node\_modules

#一些命令行工具常使用全局安装的方式

npm install -g webpack

### 4、其它命令

#更新包（更新到最新版本）

npm update 包名

#全局更新

npm update -g 包名

#卸载包

npm uninstall 包名

#全局卸载

npm uninstall -g 包名

# 06-Babel 转码器

## 简介



Babel是一个广泛使用的转码器，可以将ES6代码转为ES5代码，从而在现有环境执行执行。

这意味着，你可以现在就用 ES6 编写程序，而不用担心现有环境是否支持。

## 二、安装

### 安装命令行转码工具

Babel提供babel-cli工具，用于命令行转码。它的安装命令如下：

npm install --global babel-cli

#查看是否安装成功

babel --version

## 三、Babel的使用

### 1、初始化项目

 npm init -y

### 2、创建文件

src/example.js

下面是一段ES6代码：

// 转码前

// 定义数据

*let* input = [1, 2, 3]

// 将数组的每个元素 +1

input = input.map(*item* *=>* *item* + 1)

console.log(input)

### 2、配置.babelrc

Babel的配置文件是.babelrc，存放在项目的根目录下，该文件用来设置转码规则和插件，基本格式如下。

{

*"presets"*: [],

*"plugins"*: []

}

presets字段设定转码规则，将es2015规则加入 .babelrc：

{

*"presets"*: ["es2015"],

*"plugins"*: []

}

### 3、安装转码器

在项目中安装

npm install --save-dev babel-preset-es2015

### 4、转码

# 转码结果写入一个文件

mkdir dist1

# --out-file 或 -o 参数指定输出文件

babel src/example.js --out-file dist1/compiled.js

# 或者

babel src/example.js -o dist1/compiled.js

# 整个目录转码

mkdir dist2

# --out-dir 或 -d 参数指定输出目录

babel src --out-dir dist2

# 或者

babel src -d dist2

# 07-模块化module

## 一、模块化简介

### 1、模块化产生的背景

随着网站逐渐变成"互联网应用程序"，嵌入网页的Javascript代码越来越庞大，越来越复杂。

Javascript模块化编程，已经成为一个迫切的需求。理想情况下，开发者只需要实现核心的业务逻辑，其他都可以加载别人已经写好的模块。

但是，Javascript不是一种模块化编程语言，它不支持"类"（class），包（package）等概念，更遑论"模块"（module）了。

### 2、什么是模块化开发

传统非模块化开发有如下的缺点：

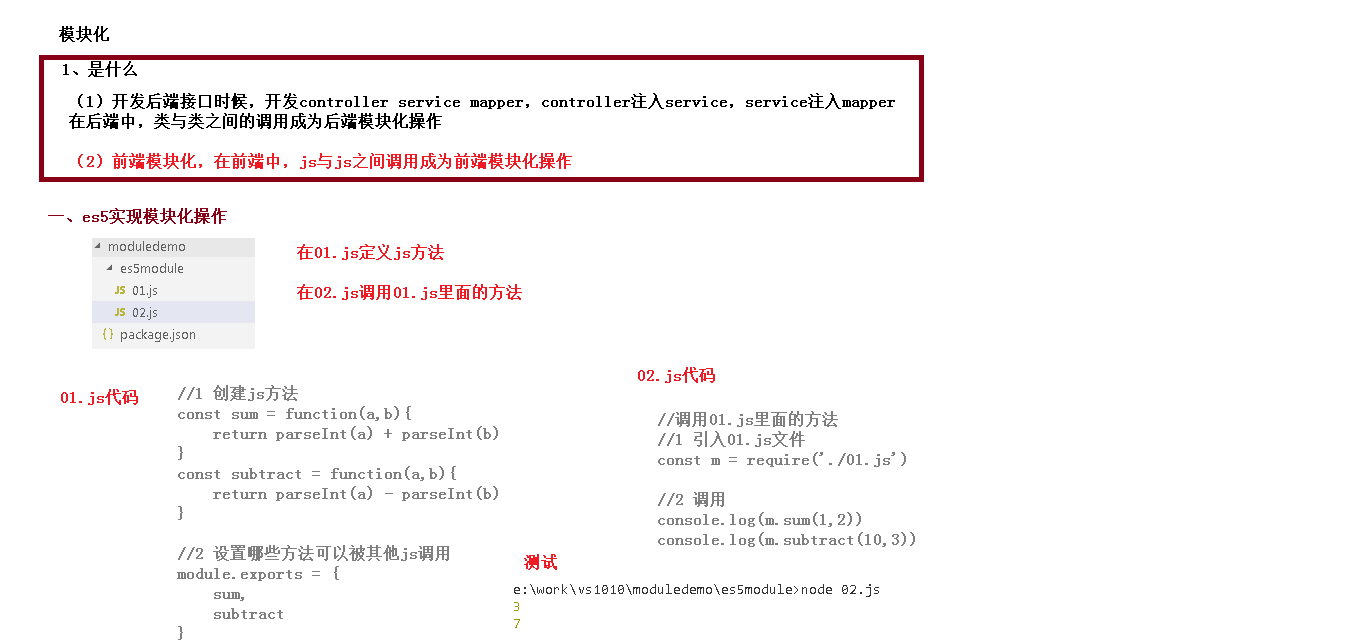
* 命名冲突
* 文件依赖

模块化规范：

* CommonJS模块化规范
* ES6模块化规范

## CommonJS模块规范

每个文件就是一个模块，有自己的作用域。在一个文件里面定义的变量、函数、类，都是私有的，对其他文件不可见。



### 1、创建“module”文件夹

### 2、导出模块*module*.*exports*

创建 common-js模块化/四则运算.js

// 定义成员：

*const* sum = *function*(*a*,*b*){

    return parseInt(*a*) + parseInt(*b*)

}

*const* subtract = *function*(*a*,*b*){

    return parseInt(*a*) - parseInt(*b*)

}

*const* multiply = *function*(*a*,*b*){

    return parseInt(*a*) \* parseInt(*b*)

}

*const* divide = *function*(*a*,*b*){

    return parseInt(*a*) / parseInt(*b*)

}

导出模块中的成员

//2 设置哪些方法可以被其他js调用

// 导出成员：

*module*.*exports* = {

    sum: sum,

    subtract: subtract,

    multiply: multiply,

    divide: divide

}

简写

//简写

*module*.*exports* = {

    sum,

    subtract,

    multiply,

    divide

}

### 3、导入模块require

创建 common-js模块化/引入模块.js

//引入模块，注意：当前路径必须写 ./

*const* m = require('./四则运算.js')

console.log(m)

*const* result1 = m.sum(1, 2)

*const* result2 = m.subtract(1, 2)

console.log(result1, result2)

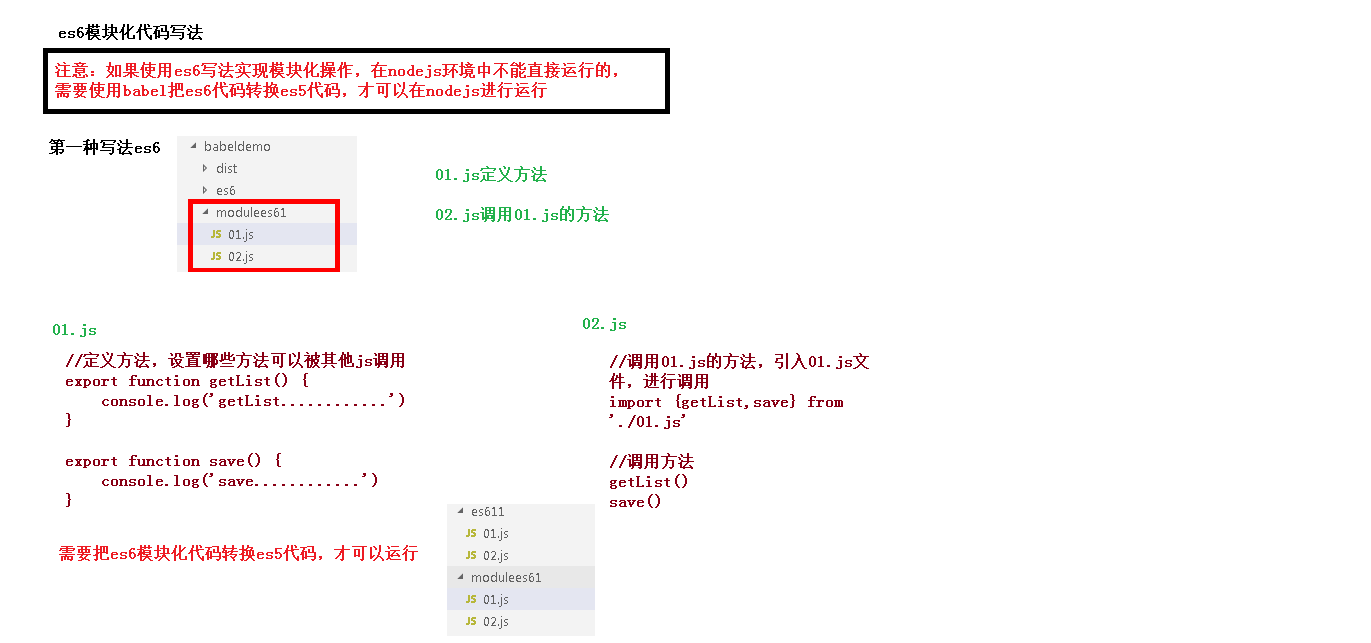
### 4、运行程序

node common-js模块化/引入模块.js

CommonJS使用 exports 和require 来导出、导入模块。

## 三、ES6模块化规范

ES6使用 export 和 import 来导出、导入模块。



### 1、导出模块export

创建 es6模块化/userApi.js

export *function* getList() {

    console.log('获取数据列表')

}

export *function* save() {

    console.log('保存数据')

}

### 2、导入模块import

创建 es6模块化/userComponent.js

//只取需要的方法即可，多个方法用逗号分隔

import { getList, save } from "./userApi.js"

getList()

save()

**注意：这时的程序无法运行的，因为ES6的模块化无法在Node.js中执行，需要用Babel编辑成ES5后再执行。**

### 3、运行程序

node es6模块化-dist/userComponent.js

## 四、ES6模块化的另一种写法



### 1、导出模块

创建 es6模块化/userApi2.js

//定义方法，设置哪些方法可以被其他js调用

export default {

    getList() {

        console.log('getList.....')

    },

    update() {

        console.log('update.....')

    }

}

### 2、导入模块

创建 es6模块化/userComponent2.js

//调用01.js的方法，引入01.js文件，进行调用

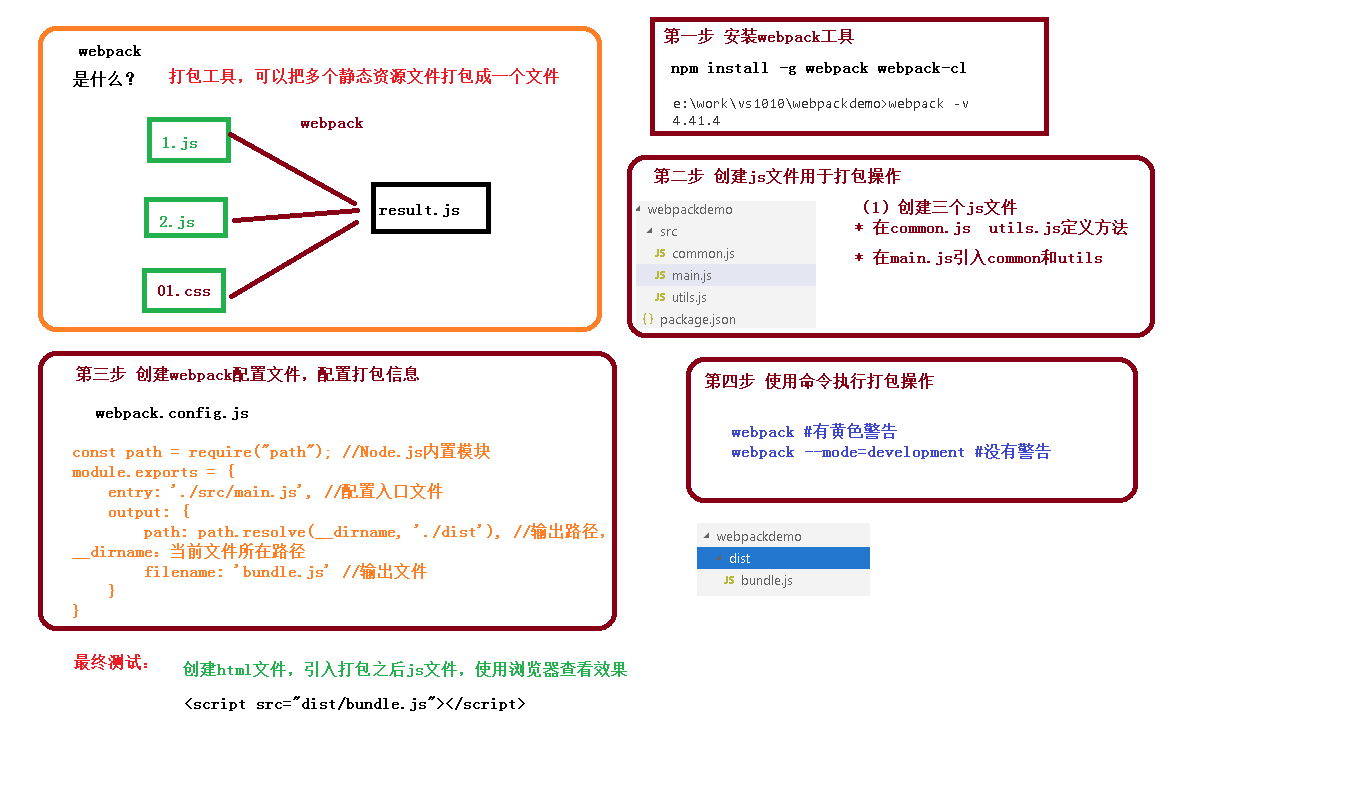
import m from './ userApi2.js'

//调用方法

m.getList()

m.update()

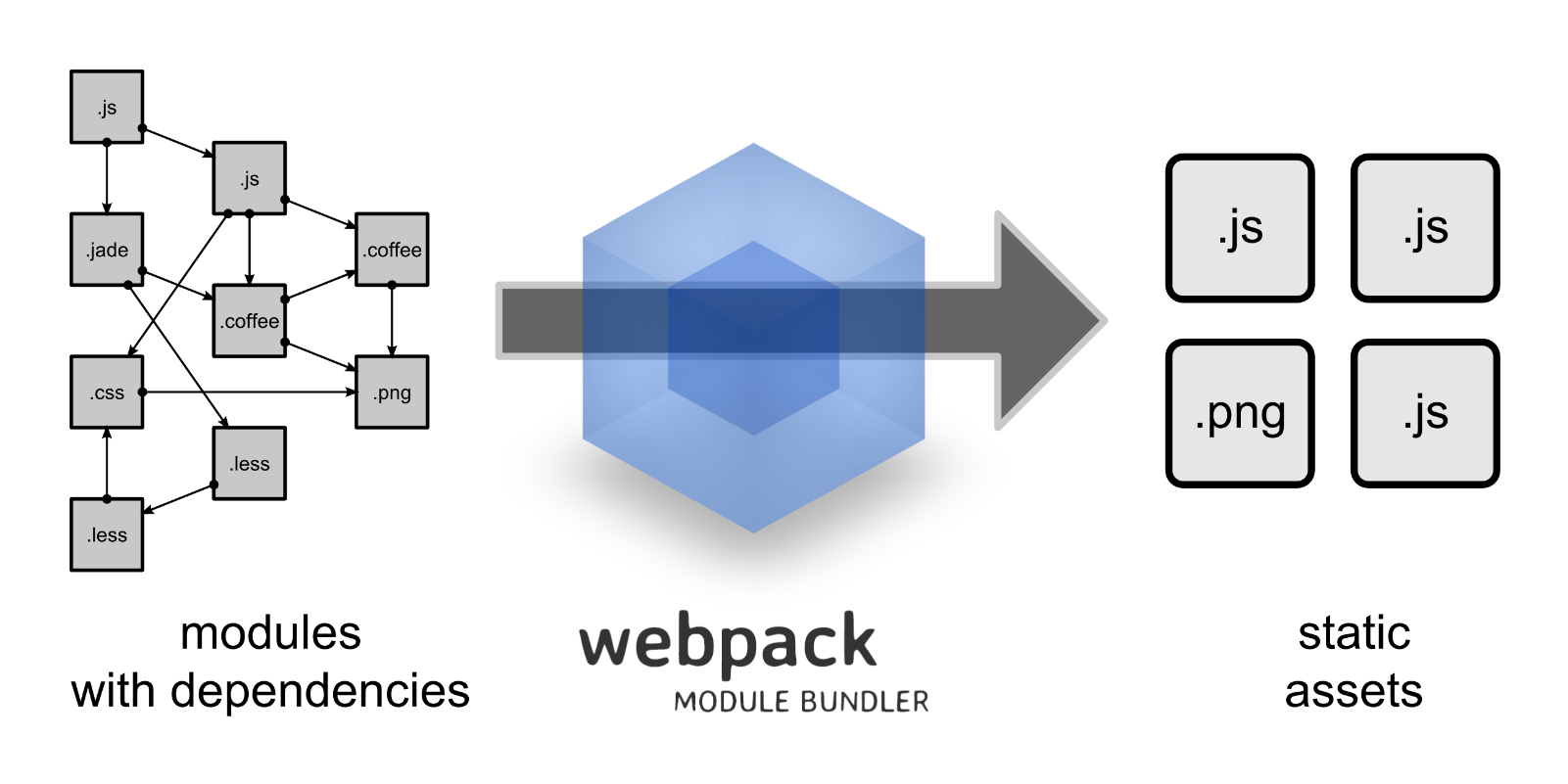
# 08-Webpack



## 一、什么是Webpack

Webpack 是一个前端资源加载/打包工具。它将根据模块的依赖关系进行静态分析，然后将这些模块按照指定的规则生成对应的静态资源。

从图中我们可以看出，Webpack 可以将多种静态资源 js、css、less 转换成一个静态文件，减少了页面的请求。



## 二、Webpack安装

### 1、全局安装

npm install -g webpack webpack-cli

### 2、安装后查看版本号

webpack -v

## 三、初始化项目

### 1、创建webpack文件夹

进入webpack目录，执行命令

npm init -y

### 2、创建src文件夹

### 3、src下创建common.js

*exports*.info = *function* (*str*) {

    document.write(*str*);//会在浏览器中输出

}

### 4、src下创建utils.js

*exports*.add = *function* (*a*, *b*) {

    return *a* + *b*;

}

### 5、src下创建main.js

*const* common = require('./common.js');

*const* utils = require('./utils.js');

common.info('Hello world!' + utils.add(100, 200));

## 四、JS打包

### 1、webpack目录下创建配置文件webpack.config.js

以下配置的意思是：读取当前项目目录下src文件夹中的main.js（入口文件）内容，分析资源依赖，把相关的js文件打包，打包后的文件放入当前目录的dist文件夹下，打包后的js文件名为bundle.js

*const* path = require("path"); //Node.js内置模块

*module*.*exports* = {

    entry: './src/main.js', //配置入口文件

    output: {

        path: path.resolve(\_\_dirname, './dist'), //输出路径，\_\_dirname：当前文件所在路径

        filename: 'bundle.js' //输出文件

    }

}

### 2、命令行执行编译命令

webpack #有黄色警告

webpack --mode=development #没有警告 development(带格式) production(一行)

#执行后查看bundle.js 里面包含了上面两个js文件的内容并惊醒了代码压缩

也可以配置项目的npm运行命令，修改package.json文件

*"scripts"*: {

*//………,*

*"dev"*: "webpack --mode=development"

  },

运行npm命令执行打包

npm run dev

### 3、webpack目录下创建index.html

引用bundle.js

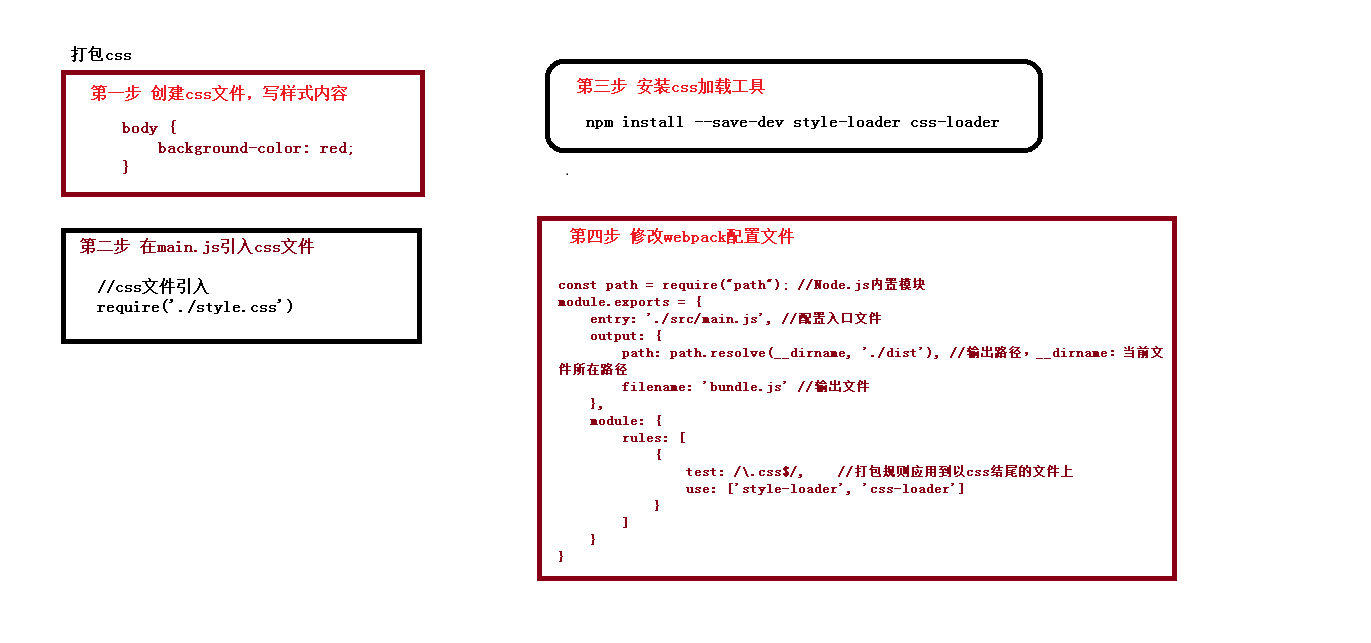
<body>

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

</body>

### 4、浏览器中查看index.html

## 五、CSS打包



### 1、安装style-loader和 css-loader

Webpack 本身只能处理 JavaScript 模块，如果要处理其他类型的文件，就需要使用 loader 进行转换。

Loader 可以理解为是模块和资源的转换器。

首先我们需要安装相关Loader插件，css-loader 是将 css 装载到 javascript；style-loader 是让 javascript 认识css

npm install --save-dev style-loader css-loader

### 2、修改webpack.config.js

*const* path = require("path"); //Node.js内置模块

*module*.*exports* = {

    . . .

    output: {

    },

    module: {

        rules: [

            {

                test: /\.css$/,    //打包规则应用到以css结尾的文件上

                use: ['style-loader', 'css-loader']

            }

        ]

    }

}

### 3、在src文件夹创建style.css

body {

*background-color*: red;

}

### 4、修改main.js

在第一行引入style.css

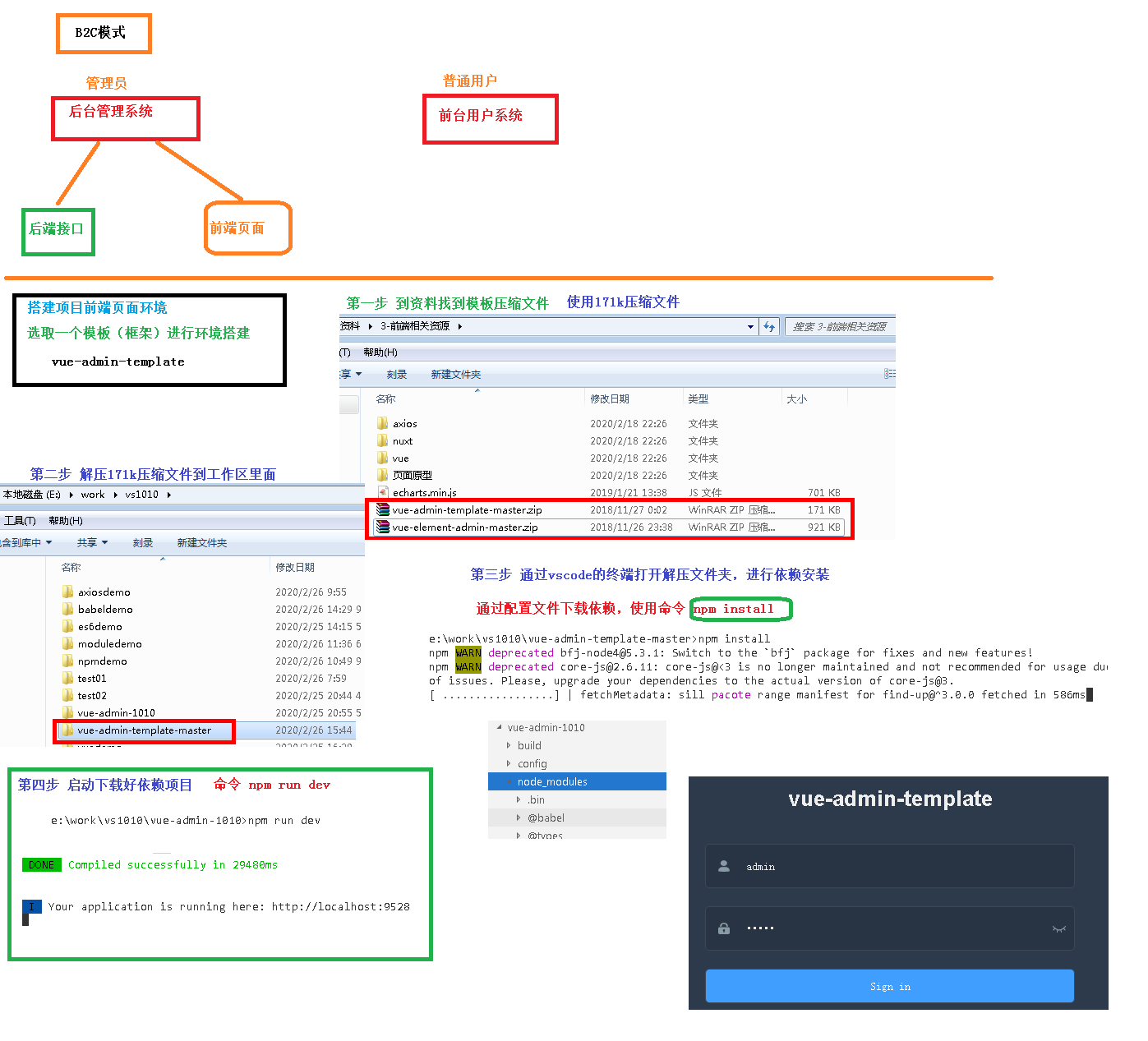
//css文件引入

require('./style.css')

### 5、浏览器中查看index.html

看看背景是不是变成粉色啦？

# 9-vue-element-admin



## 一、vue-element-admin

### 1、简介

而vue-element-admin是基于element-ui 的一套后台管理系统集成方案。

**功能：**[https://panjiachen.github.io/vue-element-admin-site/zh/guide/#功能](https://panjiachen.github.io/vue-element-admin-site/zh/guide/#%E5%8A%9F%E8%83%BD)

**GitHub地址：**<https://github.com/PanJiaChen/vue-element-admin>

**项目在线预览：**[https://panjiachen.gitee.io/vue-element-admin](https://panjiachen.gitee.io/vue-element-admin/#/login?redirect=%2Fdashboard)

### 2、安装

# 解压压缩包

# 进入目录  
cd vue-element-admin-master  
 # 安装依赖  
npm install  
 # 启动。执行后，浏览器自动弹出并访问http://localhost:9527/  
npm run dev

## 二、vue-admin-template

### 1、简介

vueAdmin-template是基于vue-element-admin的一套后台管理系统基础模板（最少精简版），可作为模板进行二次开发。

**GitHub地址：**<https://github.com/PanJiaChen/vue-admin-template>

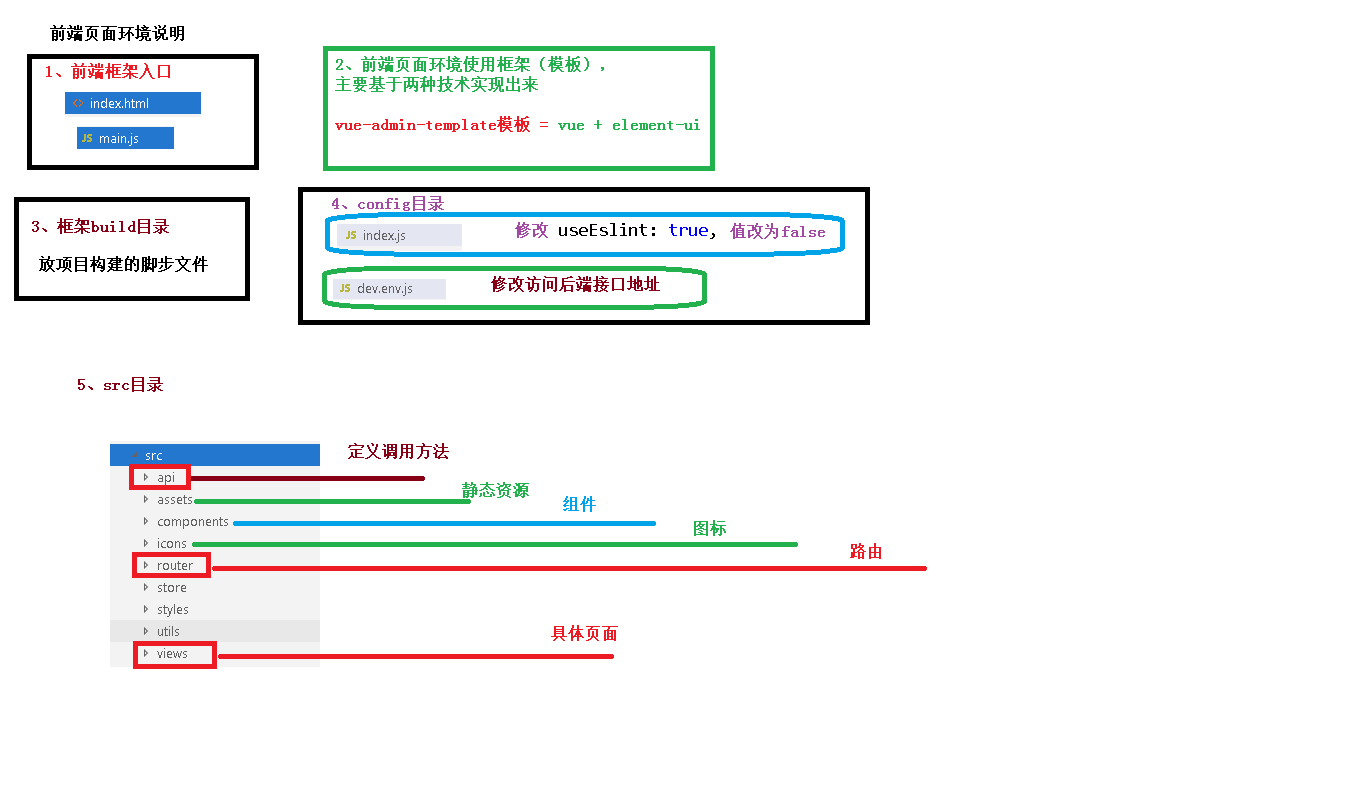
**建议：**你可以在 vue-admin-template 的基础上进行二次开发，把 vue-element-admin当做工具箱，想要什么功能或者组件就去 vue-element-admin 那里复制过来。

### 2、安装

# 解压压缩包

# 进入目录  
cd vue-admin-template-master  
# 安装依赖  
npm install  
# 启动。执行后，浏览器自动弹出并访问http://localhost:9528/  
npm run dev

# 10-后台系统前端项目创建



## 一、项目的创建和基本配置

### 1、创建项目

将vue-admin-template-master重命名为guli-admin

### 2、修改项目信息

package.json

{

"name": "guli-admin",

......

"description": "谷粒学院后台管理系统",

"author": "Helen <55317332@qq.com>",

......

}

### 3、如果需要修改端口号

config/index.js中修改

port: 9528

### 4、项目的目录结构

.

├── build // 构建脚本

├── config // 全局配置

├── node\_modules // 项目依赖模块

├── src //项目源代码

├── static // 静态资源

└── package.jspon // 项目信息和依赖配置

src

├── api // 各种接口

├── assets // 图片等资源

├── components // 各种公共组件，非公共组件在各自view下维护

├── icons //svg icon

├── router // 路由表

├── store // 存储

├── styles // 各种样式

├── utils // 公共工具，非公共工具，在各自view下维护

├── views // 各种layout

├── App.vue //\*\*\*项目顶层组件\*\*\*

├── main.js //\*\*\*项目入口文件\*\*\*

└── permission.js //认证入口

### 5、运行项目

npm run dev

## 二、登录页修改

src/views/login/index.vue（登录组件）

4行

<h3 class="title">谷粒学院后台管理系统</h3>

28行

<el-button :loading="loading" type="primary" style="width:100%;" @click.native.prevent="handleLogin">

登录

</el-button>

## 三、页面零星修改

### 1、标题

index.html（项目的html入口）

<title>谷粒学院后台管理系统</title>

修改后热部署功能，浏览器自动刷新

### 2、国际化设置

打开 src/main.js（项目的js入口），第7行，修改语言为 zh-CN，使用中文语言环境，例如：日期时间组件

import locale from 'element-ui/lib/locale/lang/zh-CN' // lang i18n

### 3、icon

复制 favicon.ico 到根目录

### 4、导航栏文字

src/views/layout/components（当前项目的布局组件）

src/views/layout/components/Navbar.vue

13行

<el-dropdown-item>

首页

</el-dropdown-item>

17行

<span style="display:block;" @click="logout">退出</span>

### 5、面包屑文字

src/components（可以在很多项目中复用的通用组件）

src/components/Breadcrumb/index.vue

38行

meta: { title: '首页' }

## 四、Eslint语法规范型检查

### 1、ESLint简介

JavaScript 是一个动态的弱类型语言，在开发中比较容易出错。因为没有编译程序，为了寻找 JavaScript 代码错误通常需要在执行过程中不断调适。

ESLint 是一个语法规则和代码风格的检查工具，可以用来保证写出语法正确、风格统一的代码。让程序员在编码的过程中发现问题而不是在执行的过程中。

### 2、语法规则

ESLint 内置了一些规则，也可以在使用过程中自定义规则。

本项目的语法规则包括：两个字符缩进，必须使用单引号，不能使用双引号，语句后不可以写分号，代码段之间必须有一个空行等。

### 3、确认开启语法检查

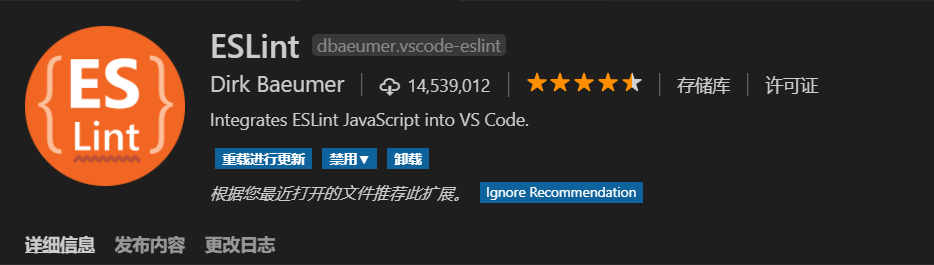
打开 config/index.js，配置是否开启语法检查

useEslint: true,

可以关闭语法检查，建议开启，养成良好的编程习惯。

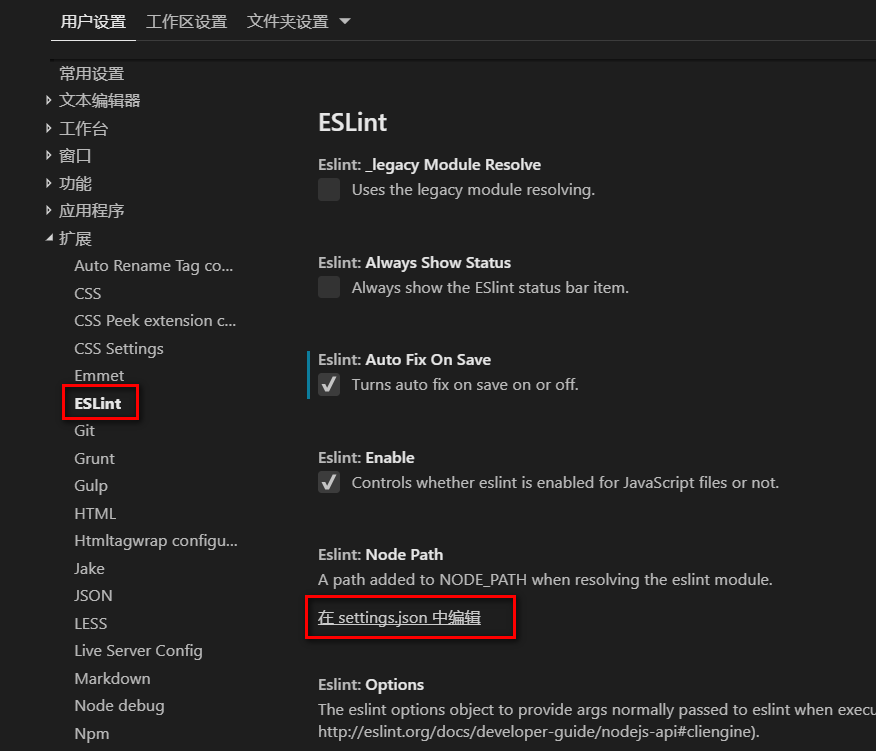
### 4、ESLint插件安装

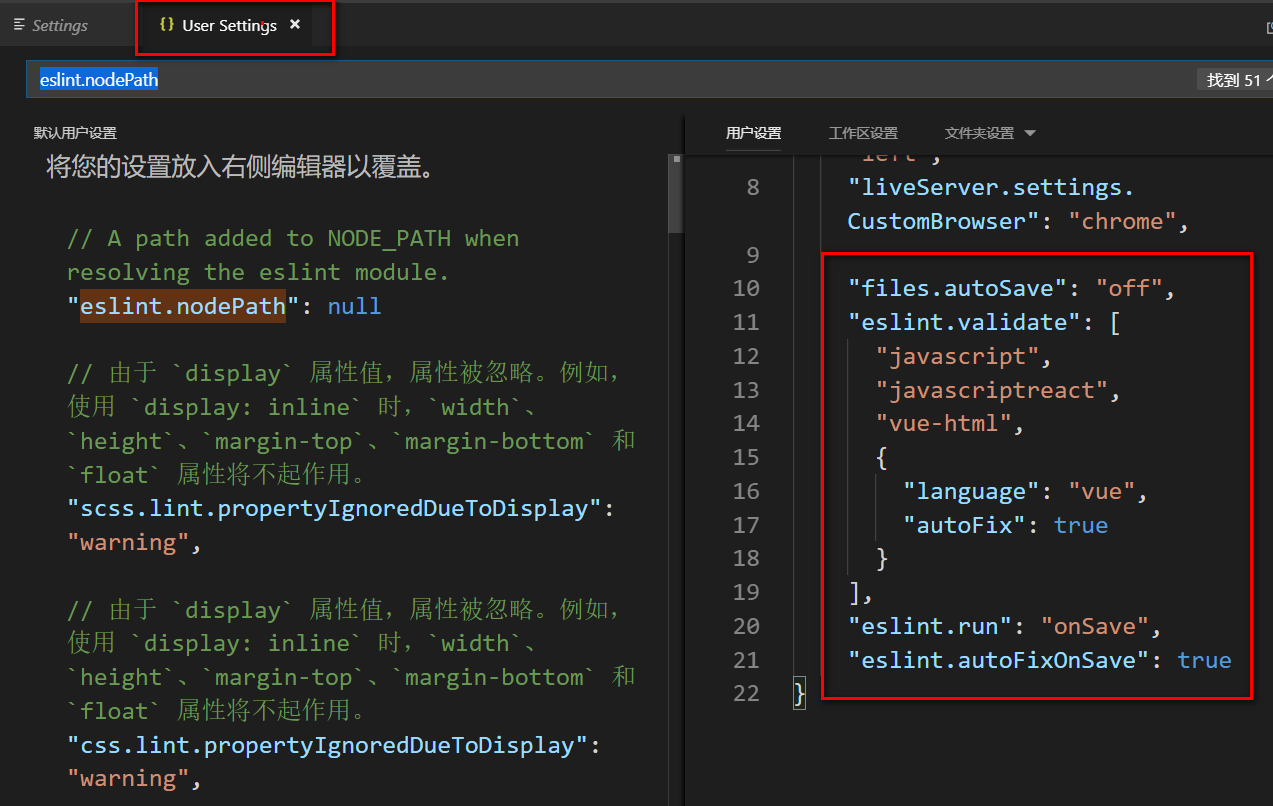
vs code的ESLint插件，能帮助我们自动整理代码格式



### 5、插件的扩展设置

选择vs code左下角的“设置”， 打开 VSCode 配置文件,添加如下配置





"files.autoSave": "off",

"eslint.validate": [

"javascript",

"javascriptreact",

"vue-html",

{

"language": "vue",

"autoFix": true

}

],

"eslint.run": "onSave",

"eslint.autoFixOnSave": true