1

2

3

# Mvvm

后端程序员眼中的前端框架设计思想。

MVVM是**Model-View-ViewModel**的简写。它本质上就是MVC 的改进版。MVVM 就是将其中的View 的状态和行为抽象化，让我们将视图 UI 和业务逻辑分开。当然这些事 ViewModel 已经帮我们做了，它可以取出 Model 的数据同时帮忙处理 View 中由于需要展示内容而涉及的业务逻辑。

模型

模型是指代表真实状态内容的领域模型（面向对象），或指代表内容的数据访问层（以数据为中心）。

视图

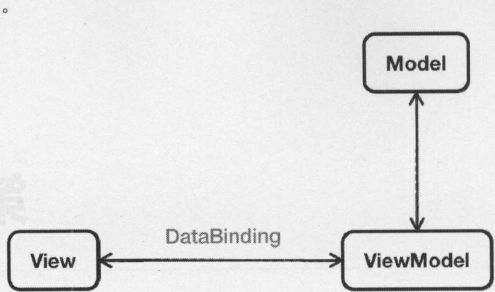
就像在MVC和MVP模式中一样，视图是用户在屏幕上看到的结构、布局和外观（UI）。

视图模型

视图模型是暴露公共属性和命令的视图的抽象。MVVM没有MVC模式的控制器，也没有MVP模式的presenter，有的是一个绑定器。在视图模型中，绑定器在视图和数据绑定器之间进行通信。

绑定器

声明性数据和命令绑定隐含在MVVM模式中。在Microsoft解决方案堆中，绑定器是一种名为XAML的标记语言。绑定器使开发人员免于被迫编写样板式逻辑来同步视图模型和视图。在微软的堆之外实现时，声明性数据绑定技术的出现是实现该模式的一个关键因素。



View绑定到ViewModel，然后执行一些命令再向它请求一个动作。而反过来，ViewModel跟Model通讯，告诉它更新来响应UI。这样便使得为应用构建UI非常的容易。

**优点**

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

1. 低耦合。视图（View）可以独立于Model变化和修改，一个ViewModel可以绑定到不同的"View"上，当View变化的时候Model可以不变，当Model变化的时候View也可以不变。

2. 可重用性。你可以把一些视图逻辑放在一个ViewModel里面，让很多view重用这段视图逻辑。

3. 独立开发。开发人员可以专注于业务逻辑和数据的开发（ViewModel），设计人员可以专注于页面设计，使用Expression Blend可以很容易设计界面并生成xaml代码。

4. 可测试。界面素来是比较难于测试的，而现在测试可以针对ViewModel来写。

# 问题

在工作中，如何快速学习一门新技术？？？？

一开始理论一定要熟读，应用可以似是而非，一知半解，能实现功能就行。

# Vue

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](https://cn.vuejs.org/v2/guide/single-file-components.html)**以及各种[**支持类库**](https://github.com/vuejs/awesome-vue#libraries--plugins)结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。

# 工具

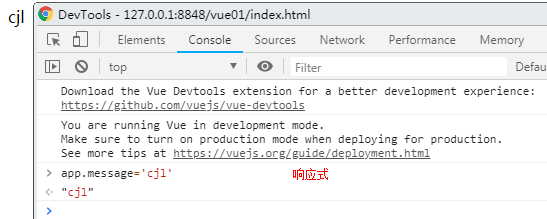
<http://www.dcloud.io/hbuilderx.html>

# 入门

官网：https://cn.vuejs.org/v2/guide/index.html

下载：<https://vuejs.org/js/vue.js>

## 声明式渲染



|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <title>vue-start</title>  <script src="js/vue.js"> </script>  </head>  <body>  <div id="app">  {{ message }}  </div>  <div id="app-2">  <span v-bind:title="message">  鼠标悬停几秒钟查看此处动态绑定的提示信息！  </span>  </div>  <script>  var app = new Vue({  el: '#app',  data: {  message: '欢迎！！'  }  })  var app2 = new Vue({  el: '#app-2',  data: {  message: '页面加载于 ' + new Date().toLocaleString()  }  })  </script>  </body>  </html> |

## 实例

实例生命周期钩子

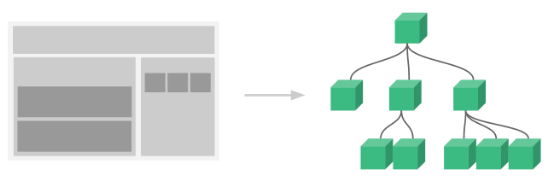


|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8">  <title>object</title>  <script src="js/vue.js"> </script>  </head>  <body>  <div id="app">  <span @click='foo'>{{msg}}</span>  <br />  <span>年龄：{{age}}</span>  </div>  <div id="ap2">      </div>  <script>  //viewmodel  let app = new Vue({  //特殊字段  el:'#app',  //字段  data:{  msg:'message',  age : 20,  students: []  },  //方法  methods:{  foo : function(){  alert('foo')  },  showMsg:function(){  alert(this.msg)  }  },  //生命周期  created:function(){  //用来初始化数据,一般是后台数据  // 不能访问元素对象的  console.log('created')  },  mounted:function(){  //完成view的元素挂接  console.log('mounted')  }  })  </script>  </body>  </html> |

## 条件与循环

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8">  <title>if-for</title>  <script src="js/vue.js"> </script>  </head>  <body>  <div id="app-3">  <p v-if="seen">现在你看到我了</p>  </div>  <div id="app-4">  <p v-if="abc">我是if</p>  <p v-else>我是else</p>  </div>  <div id="app-5">  <ol>  <li v-for="obj in arrys">  {{obj.text}}  </li>  </ol>  </div>  <script>  var app3 = new Vue({  el: '#app-3',  data: {  seen: true  }  })  var app3 = new Vue({  el: '#app-4',  data: {  abc: false  }  })  var app5 = new Vue({  el: '#app-5',  data: {  arrys: [{  text: '学习 JavaScript'  },  {  text: '学习 Vue'  },  {  text: '整个牛项目'  },  {  text: '不要睡觉'  }  ]  }  })  </script>  </body>  </html> |

## 组件化应用构建



# 组件

## 基本示例

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8">  <title>222</title>  <script src="js/vue.js"></script>  </head>  <body>  <div id="com2">  <button-counter></button-counter>  <button-counter></button-counter>  <button-counter></button-counter>  </div>  <script>  Vue.component('button-counter',{  //父属性  props:['abc'],  data:function(){  return {  //组件自己的属性  count:0  }  },  template:'<button @click="count++">点击了{{count}}次</button>'  })    //实例化对象  new Vue({  el:'#com2'  })  </script>  </body>  </html> |

## 注册

因为组件是可复用的 Vue 实例，所以它们与 new Vue 接收相同的选项，例如 data、computed、watch、methods 以及生命周期钩子等。仅有的例外是像 el这样根实例特有的选项。

**every component must have a single root element (每个组件必须只有一个根元素)**。

注意**局部注册的组件在其子组件中*不可用***。例如，如果你希望 ComponentA 在 ComponentB 中可用，则你需要这样写：

var ComponentA = { /\* ... \*/ }

var ComponentB = {

components: {

'component-a': ComponentA

},

// ...

}

或者如果你通过 Babel 和 webpack 使用 ES2015 模块，那么代码看起来更像：

import ComponentA from './ComponentA.vue'

export default {

components: {

ComponentA

},

// ...

}

注意在 ES2015+ 中，在对象中放一个类似 ComponentA 的变量名其实是 ComponentA: ComponentA 的缩写，即这个变量名同时是：

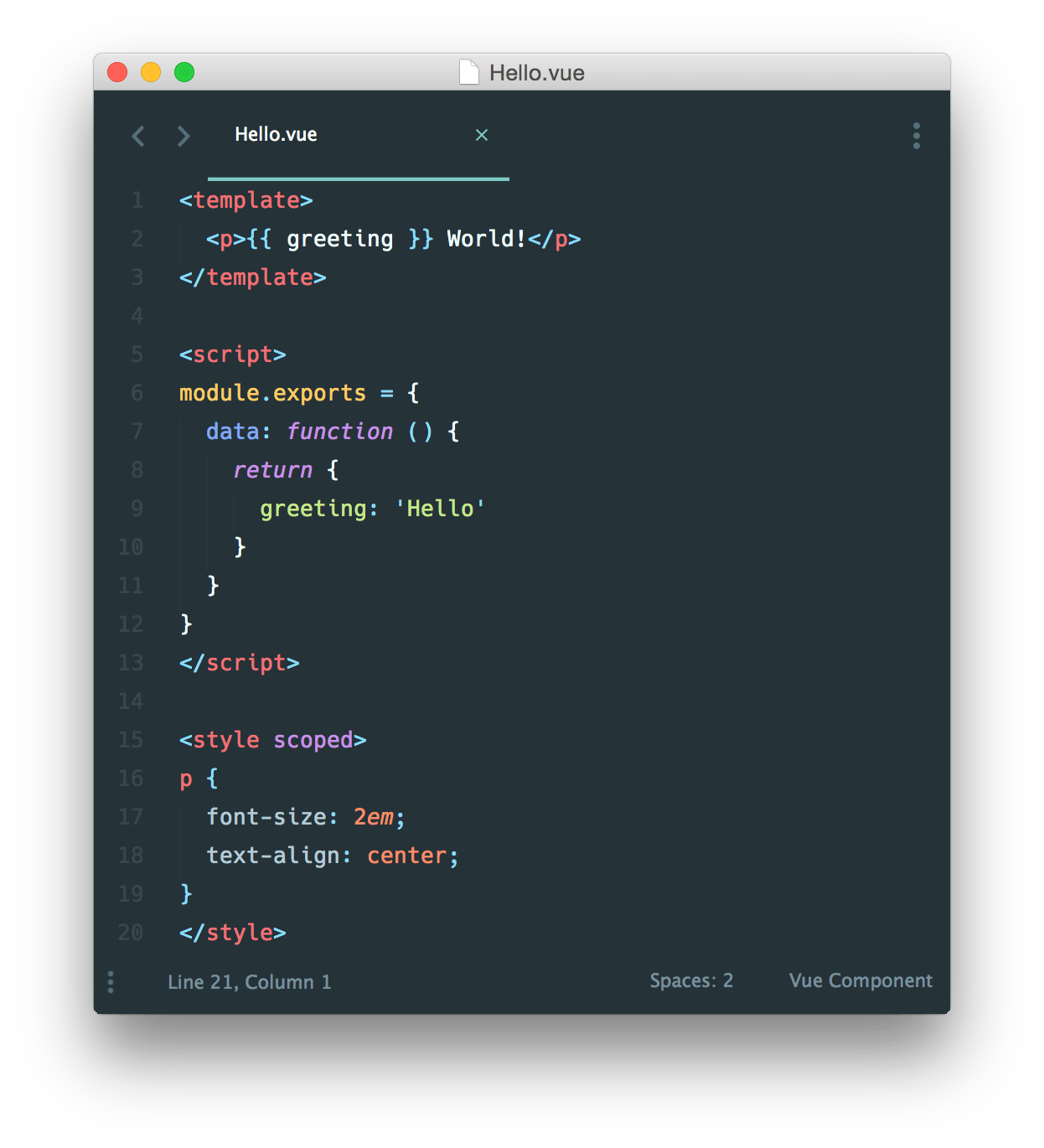
* 用在模板中的自定义元素的名称
* 包含了这个组件选项的变量名

# 单文件

使用原生的vue组件缺点：

* **全局定义 (Global definitions)** 强制要求每个 component 中的命名不得重复
* **字符串模板 (String templates)** 缺乏语法高亮，在 HTML 有多行的时候，需要用到丑陋的 \
* **不支持 CSS (No CSS support)** 意味着当 HTML 和 JavaScript 组件化时，CSS 明显被遗漏
* **没有构建步骤 (No build step)** 限制只能使用 HTML 和 ES5 JavaScript, 而不能使用预处理器，如 Pug (formerly Jade) 和 Babel

单文件好处：



## [针对刚接触 JavaScript 模块开发系统的用户](https://cn.vuejs.org/v2/guide/single-file-components.html#%E9%92%88%E5%AF%B9%E5%88%9A%E6%8E%A5%E8%A7%A6-JavaScript-%E6%A8%A1%E5%9D%97%E5%BC%80%E5%8F%91%E7%B3%BB%E7%BB%9F%E7%9A%84%E7%94%A8%E6%88%B7)

有了 .vue 组件，我们就进入了高级 JavaScript 应用领域。如果你没有准备好的话，意味着还需要学会使用一些附加的工具：

* **Node Package Manager (NPM)**：阅读 [**Getting Started guide**](https://docs.npmjs.com/getting-started/what-is-npm) 直到 10: Uninstalling global packages章节。
* **Modern JavaScript with ES2015/16**：阅读 Babel 的 [**Learn ES2015 guide**](https://babeljs.io/docs/learn-es2015/)。你不需要立刻记住每一个方法，但是你可以保留这个页面以便后期参考。

在你花一天时间了解这些资源之后，我们建议你参考 **[Vue CLI 3](https://cli.vuejs.org/zh/" \t "_blank)**。只要遵循指示，你就能很快地运行一个带有 .vue 组件、ES2015、webpack 和热重载 (hot-reloading) 的 Vue 项目!

## [针对高级用户](https://cn.vuejs.org/v2/guide/single-file-components.html#%E9%92%88%E5%AF%B9%E9%AB%98%E7%BA%A7%E7%94%A8%E6%88%B7)

CLI 会为你搞定大多数工具的配置问题，同时也支持细粒度自定义**[配置项](https://cli.vuejs.org/zh/config/" \t "_blank)**。

有时你会想从零搭建你自己的构建工具，这时你需要通过 **[Vue Loader](https://vue-loader.vuejs.org/zh/" \t "_blank)** 手动配置 webpack。关于学习更多 webpack 的内容，请查阅**[其官方文档](https://webpack.js.org/configuration/" \t "_blank)** 和 **[Webpack Academy](https://webpack.academy/p/the-core-concepts" \t "_blank)**。

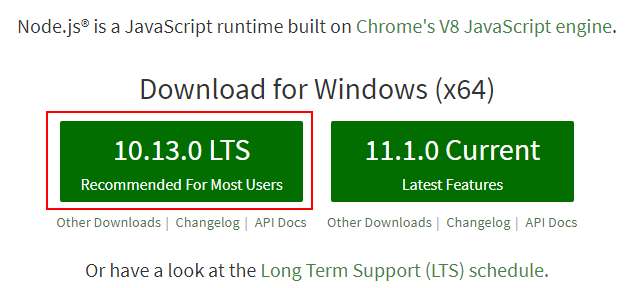
# Node.js

中文版文档：<https://nodejs.org/zh-cn/docs/>

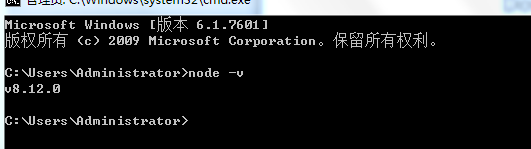
这里需要深入学习的同学可以自己课后再去仔细专研，我们后台服务器技术还是以java为主，时间有限，这里大家只要粗略知道，node是一个用js写的web服务器。前端开发人员不熟悉其他后端开发使用的服务器的情况下，也可以调试、运行前端项目，当然也可以作为生产服务器来使用，就像大家使用tomcat一样。

Node提供了解释、运行js的环境，仅此而已！！你可以认为它是java世界里的java虚拟机，事实上它的很多设计和jvm很相似，而且它的名字也叫虚拟机，这一点大家在前面学习js调试代码的时候，不知有没有注意。

下载、安装：<https://nodejs.org/en/>



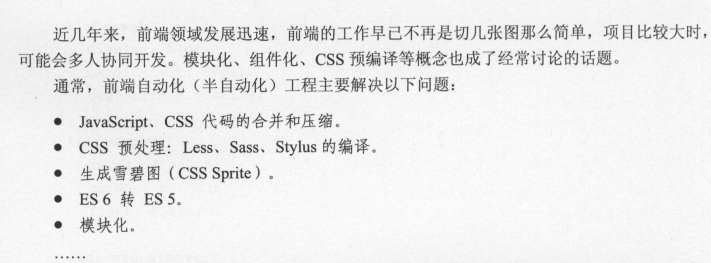
下载后直接双击安装就行，然后打开命令行，输入命令测试：



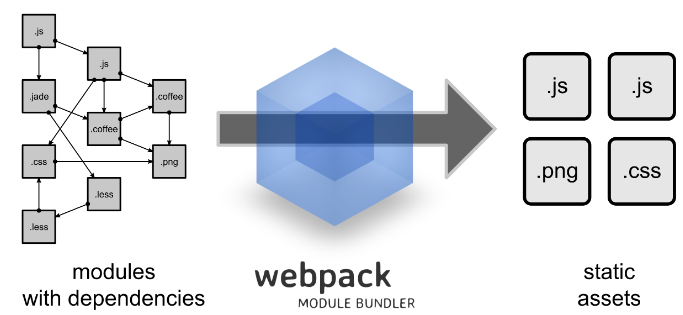
**我安装的是8.12.0这个版本，这个版本可以在提供的资源包里获取。**

那么光有个环境是不够的，大家都知道，我们在做java开发的时候，只是有个jvm是不行的，还得有包管理软件例如maven。那么我们node的包管理软件就是：**npm(node package manager)，很强大，类似于maven，其实从概念上讲它就是一个东西，只不过我们更习惯java而已。**

那么有了npm之后还是不行的，为什么呢？？原因大家看图，字太多，写不动了。



总结一句话：就是项目大了，需要模块化了，当然不能什么事都手工操作了，需要自动化提取构建资源，那么自动化总歹有个工具吧，就是让大家头疼的：**webpack。它干的活就是上图列出的问题。大家看一张官网的图吧：<https://webpack.github.io/>**



**为什么我说会让大家头疼呢？是因为这个东西相对于上面2个概念，需要我们掌握更多的东西，但是告诉大家一个好消息，这里你并不需要知道太多，你只要知道它的存在就行了。后面它的配置在vue的开发工具集成了，你根本感知不到它的存在。有兴趣的同学可以去官网了解更多的内容：<https://webpack.js.org/>**

好了，到这里大家知道大前端开发三大武器：**nodejs + npm + webpack**

# Vue-cli

Vue CLI 是一个基于 Vue.js 进行快速开发的完整系统，提供：

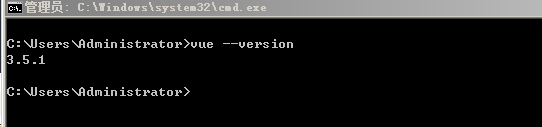
* 通过 @vue/cli 搭建交互式的项目脚手架。
* 通过 @vue/cli + @vue/cli-service-global 快速开始零配置原型开发。
* 一个运行时依赖 (@vue/cli-service)，该依赖：
  + 可升级；
  + 基于 webpack 构建，并带有合理的默认配置；
  + 可以通过项目内的配置文件进行配置；
  + 可以通过插件进行扩展。
* 一个丰富的官方插件集合，集成了前端生态中最好的工具。
* 一套完全图形化的创建和管理 Vue.js 项目的用户界面。

Vue CLI 致力于将 Vue 生态中的工具基础标准化。它确保了各种构建工具能够基于智能的默认配置即可平稳衔接，这样你可以专注在撰写应用上，而不必花好几天去纠结配置的问题。与此同时，它也为每个工具提供了调整配置的灵活性，无需 eject。

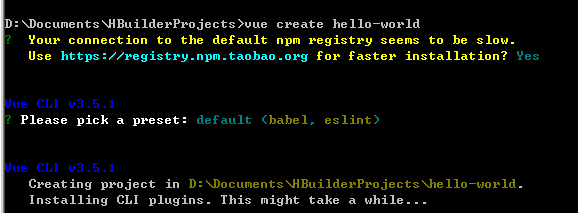
****

## 安装

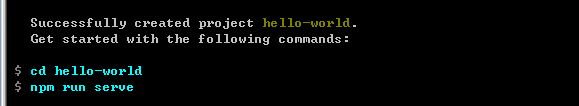
cnpm install -g @vue/cli

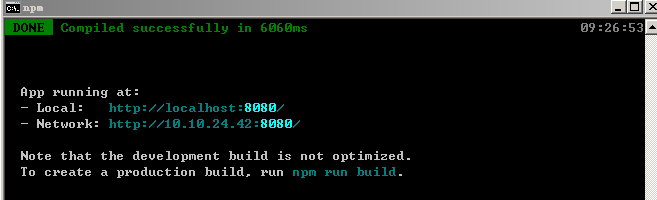


## 创建项目

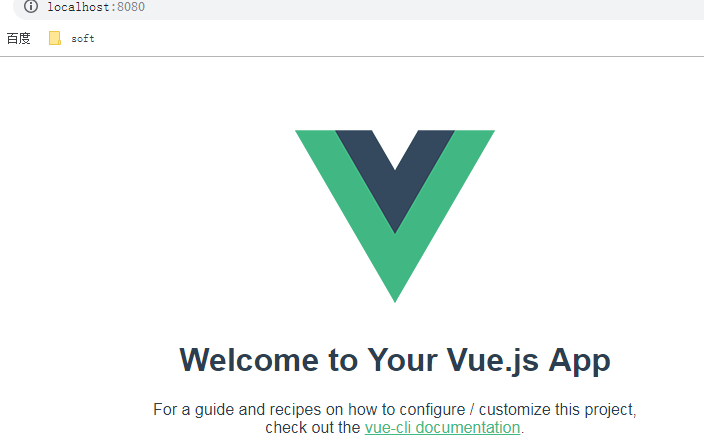


## 运行

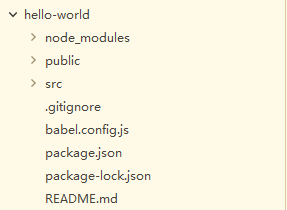




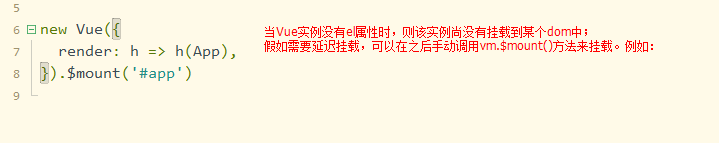
结果



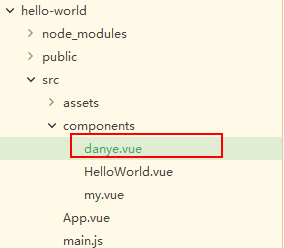
## 项目结构



入口代码main.js

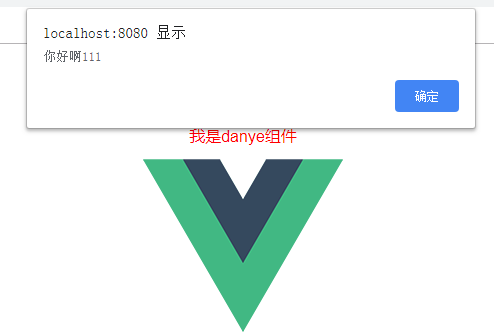


# 手写单文件



|  |
| --- |
| <template>  <div>  <span @click="fun01" class="red">{{content}}</span>  </div>  </template>  <script>  export default {  props:{msg:{type:String,default:'jfjdj'},count:{type:Number,default:111}},  data(){  return {  content:'我是danye组件'  }  },  methods:{  fun01(){  alert(this.msg+this.count)  }  }  }  </script>  <style scoped="scoped">  .red{  color: red;  }  </style> |

效果

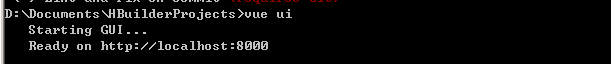


# Iview

<https://www.iviewui.com/docs/guide/start>



使用界面创建新项目





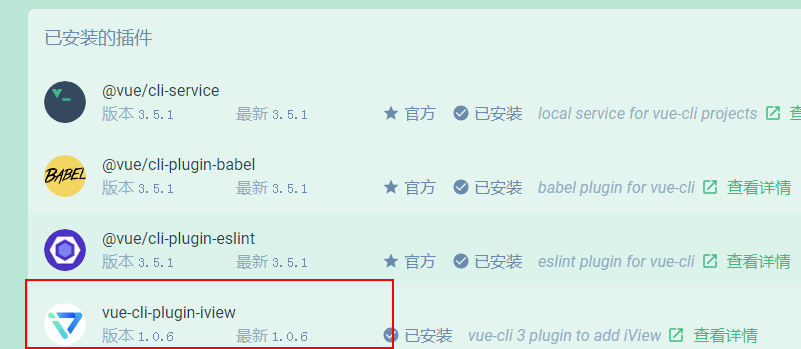
创建iview-demo成功后



安装vue-cli-plugin-iview插件



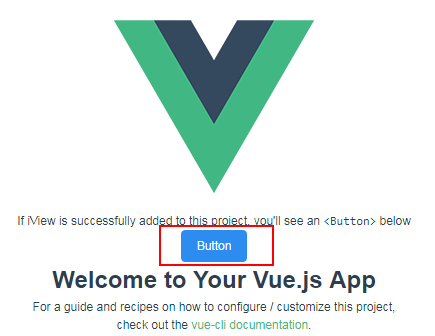
安装成功



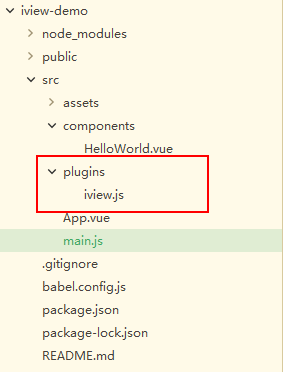
启动项目



启动成功



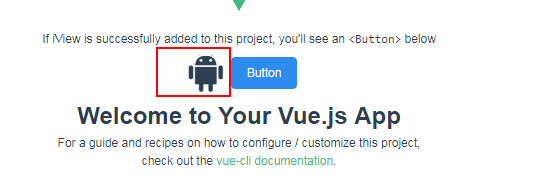
项目文档结构



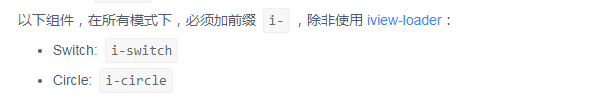
练习：添加一个图标

<Icon type="logo-android" size="50"/>

效果







# Iview-admin

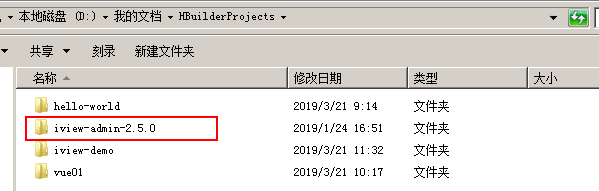
开发后台管理系统的开源前端模板工程

<https://github.com/iview/iview-admin>

直接下载最新发布版源码包



解压到HBProjects目录下



使用说明



安装依赖包

考虑下载速度，可以换为cnpm，但是会有莫名其妙的编译问题，尼玛！！！

建议还是使用npm

但是如果编译有错误时请不要使用使用cnpm，尤其是新版本node，请卸载cnpm，使用npm安装依赖包。

**npm uninstall cnpm –g**

# 分类管理

## Catalog.vue

使用树形组件，参考iview-tree

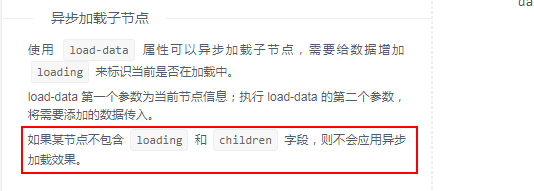
|  |
| --- |
| <template>  <Tree :data="data2" show-checkbox></Tree>  </template>  <script>  export default {  data () {  return {  data2: [  {  title: 'parent 1',  expand: true,  children: [  {  title: 'parent 1-1',  expand: true,  children: [  {  title: 'leaf 1-1-1'  },  {  title: 'leaf 1-1-2'  }  ]  },  {  title: 'parent 1-2',  expand: true,  children: [  {  title: 'leaf 1-2-1'  },  {  title: 'leaf 1-2-1'  }  ]  }  ]  }  ]  }  }  }  </script> |

## Router.js

路由中添加新组件

|  |
| --- |
| {  path: '/catalog',  name: 'catalog',  meta: {  hideInBread: true  },  component: Main,  children: [  {  path: 'catalog\_page',  name: '分类管理',  meta: {  icon: 'md-planet',  title: '分类管理'  },  component: () => import('@/view/catalog/catalog.vue')  }  ]  }, |

## 动态加载



|  |
| --- |
| <template>  <div>  <Tree :data="data2" show-checkbox></Tree>  <!-- 动态加载数据 -->  <Tree :data="data3" :load-data="loadData" show-checkbox></Tree>  </div>  </template>  <script>  export default {  data () {  return {  data2: [  {  title: 'parent 1',  expand: true,  children: [  {  title: 'parent 1-1',  expand: true,  children: [  {  title: 'leaf 1-1-1'  },  {  title: 'leaf 1-1-2'  }  ]  },  {  title: 'parent 1-2',  expand: true,  children: [  {  title: 'leaf 1-2-1'  },  {  title: 'leaf 1-2-1'  }  ]  }  ]  }  ],  data3: [  {  title: 'parent',  loading: false,  children: []  }  ]  }  },  methods:{  loadData (item, callback) {  setTimeout(() => {  const data = [  {  title: 'children',  //loading: true,  //children: []  },  {  title: 'children',  loading: false,  children: []  }  ];  callback(data);  }, 1000);  }  }  }  </script> |

## 远程加载

Catalog.js

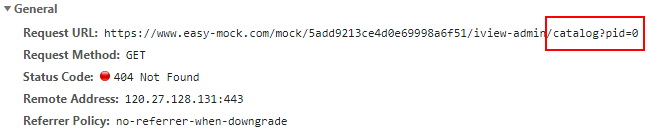
|  |
| --- |
| import axios from '@/libs/api.request'  export const getCatalogByPid = (pid) => {  return axios.request({  url: 'catalog',  params:{pid},  method: 'get'  })  } |

导入包

|  |
| --- |
| import {getCatalogByPid} from '../../api/catalog.js'  //远程加载  loadCatalog(item,callback){  getCatalogByPid(item.id).then(  res => {  console.log(res)  callback(res.data);  }  )  } |

默认调用mock方法：



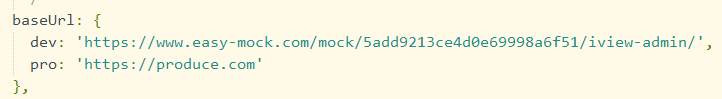


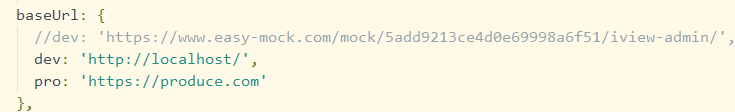
改成调用后端api的url

查看该文件

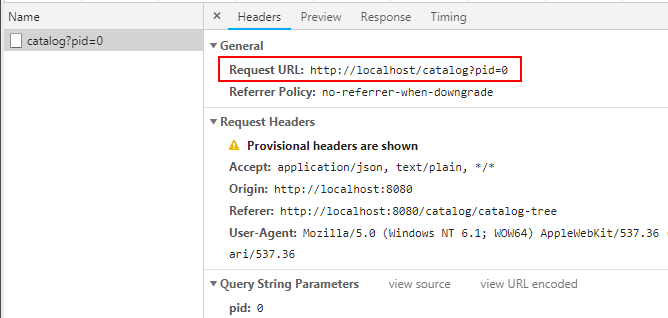


修改基础路径就可以





再次调用，变成了开发服务器的URL



这时候开发就顺理成章地到了后端服务器开发阶段了，让我们回归java，哈哈。

下节课再见！！！！！