# Vue\_05

# 目标

* 定义组件实现父子组件间传值(边界处理)
* 能够使用插槽对组件复用

# 一、vue组件

## 1.1、概述

组件 (Component) 是 Vue.js 最强大的功能之一，它是html、css、js等的一个聚合体。

组件化

* 将一个具备完整功能的项目的一部分分割多处使用
* 加快项目的进度
* 可以进行项目的复用

组件注册分为：全局注册和局部注册

## 1.2、全局注册

Vue.component('组件名称', { })第1个参数是标签名称，第2个参数是一个选项对象。全局组件注册后，任何vue实例都可以用。

**组件注意事项：**

* 构造 Vue 实例时传入的各种选项大多数都可以在组件里使用，只有一个例外：data必须是函数，同时这个函数要求返回一个对象
* 组件模板必须是单个根元素
* 组件模板的内容可以是模板字符串

Vue.component('HelloWorld', {

data: function(){

return {

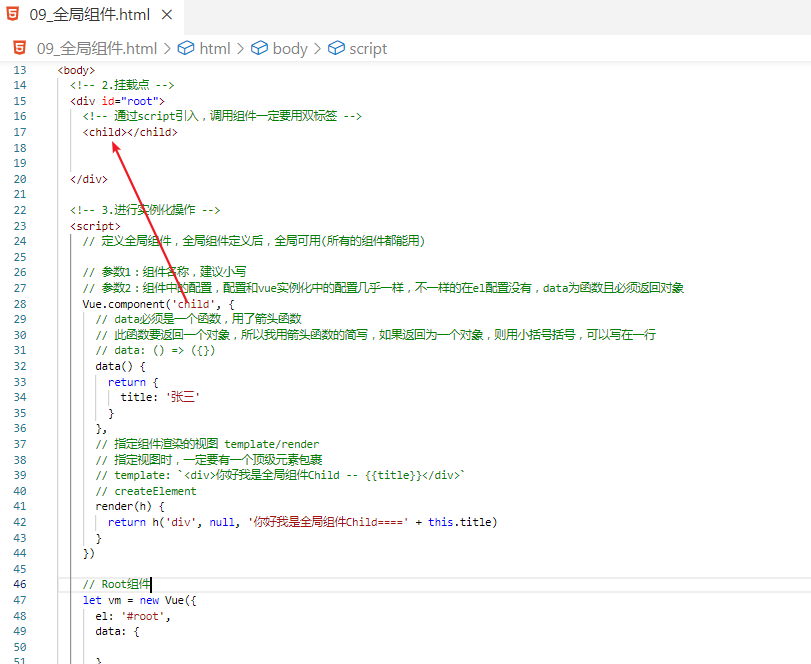
msg: 'HelloWorld'

}

},

template: '<div>{{msg}}</div>'

});



## 1.3、局部注册

只能在当前注册它的vue实例中使用，通过某个 Vue 实例/组件的实例选项 components 注册仅在其作用域中可用的组件

var Child = {

template: '<div>A custom component!</div>'

}

new Vue({

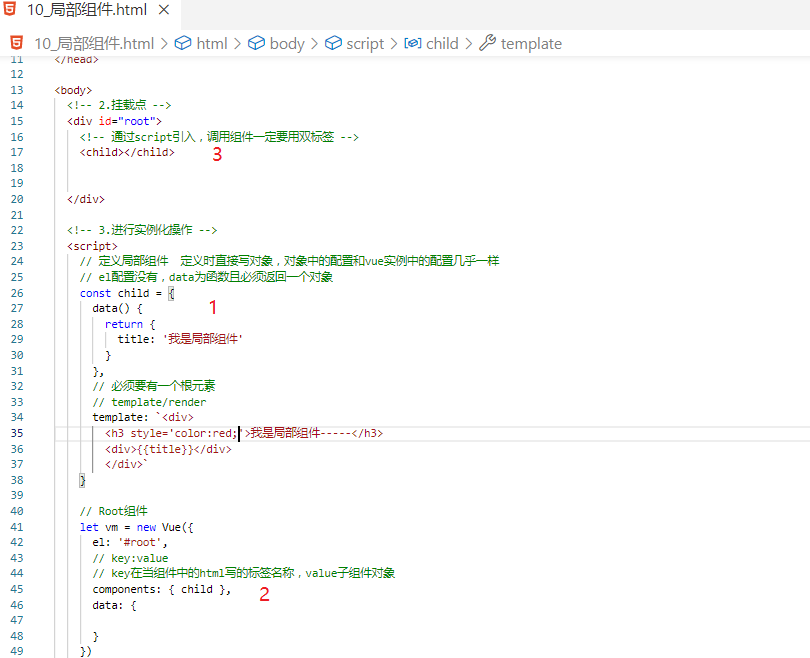
components: {

// <my-component> 将只在父组件模板中可用

'my-component': Child

}

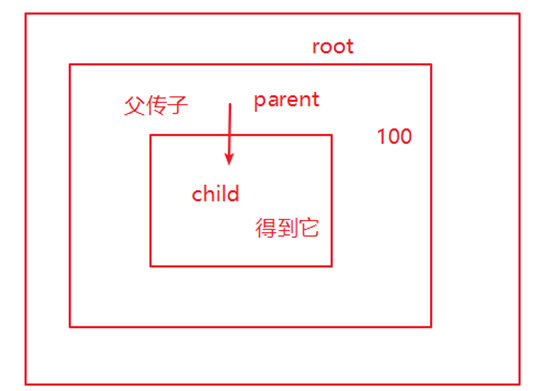
})

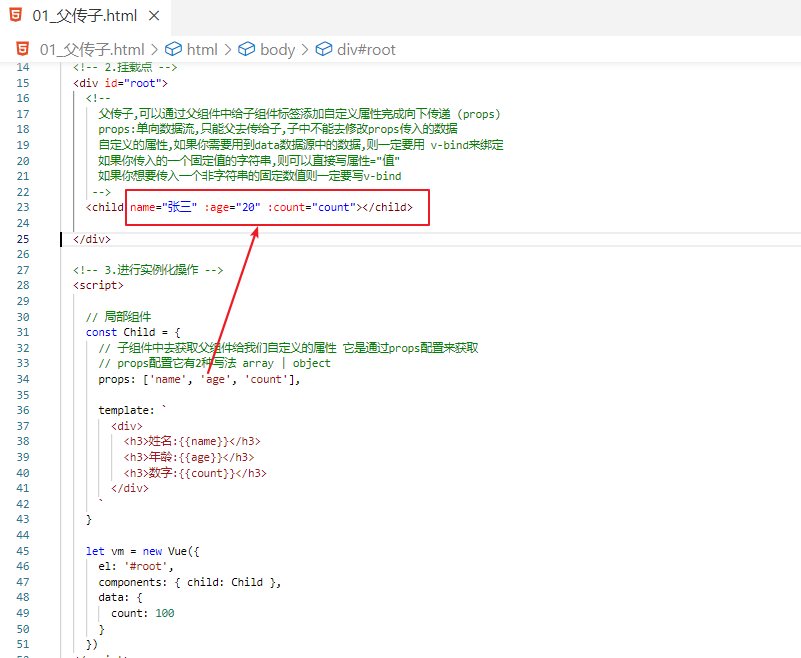


## 1.4、组件间传值

### 1.4.1、父组件向子组件传值

* 父组件以属性的形式绑定值到子组件身上
* 子组件通过使用属性props接收(props是单向绑定的(只读属性)：当父组件的属性变化时，将传导给子组件，但是反过来不会)





props对象写法

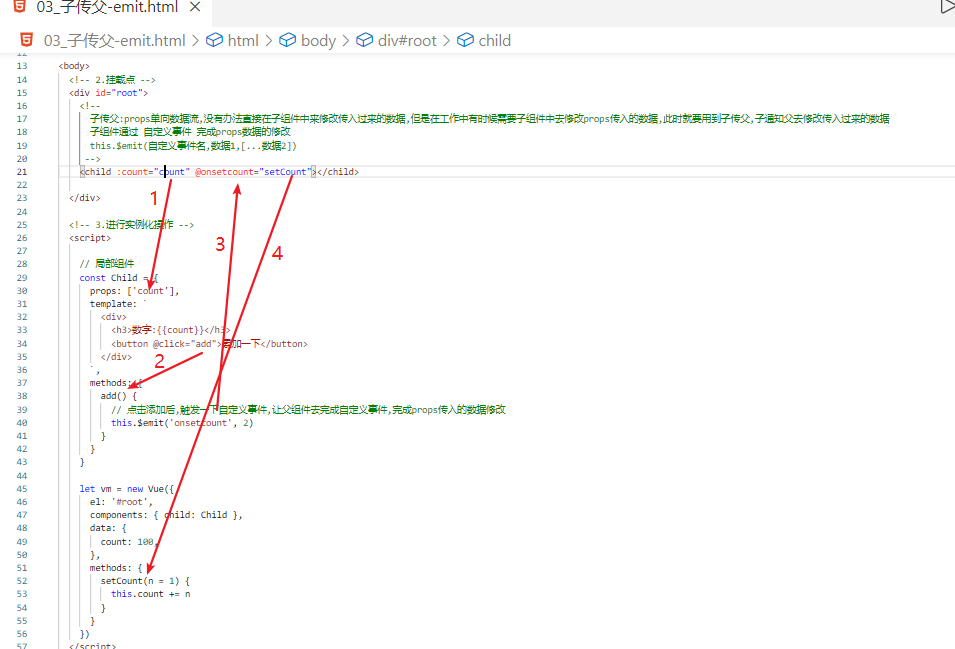


总结一下props定义方案

* 如果你定义的组件只是你当前自己界面所用，可以把props定义数组
* 如果你定义的组件给自己和别人一起使用，则一定要用对象来限制
* props如果用对象的方式来限制类型，则类型字母**一定要大写**
* props中传入的数据可以为任何类型

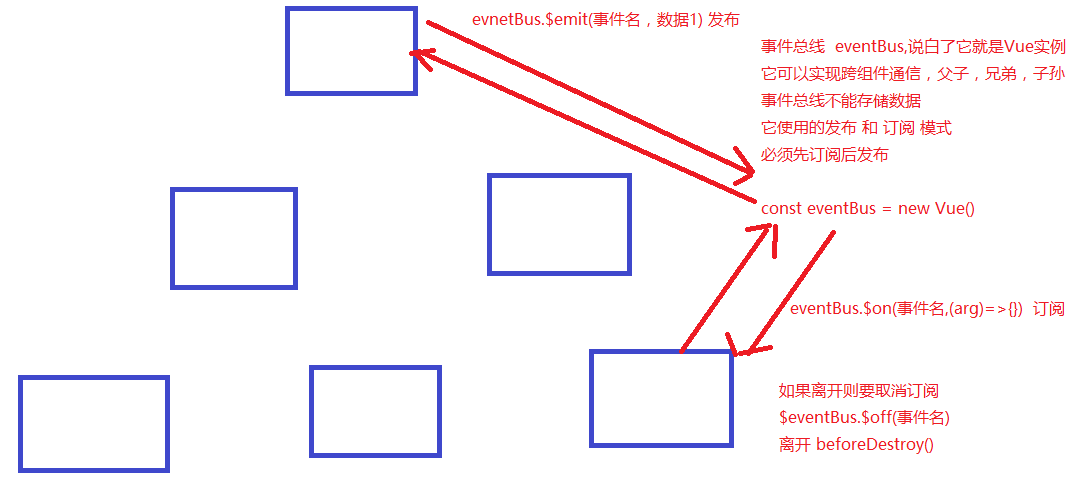
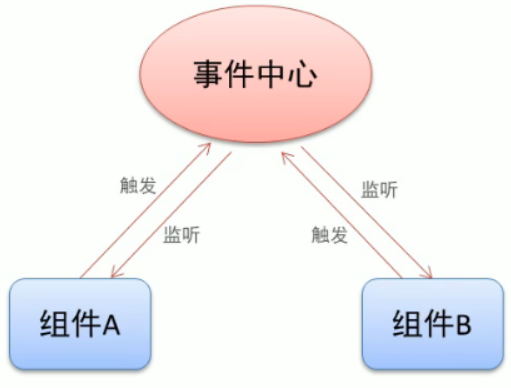
### 1.4.2、子组件向父组件传值

* 子组件用$emit()定义自定义事件，$emit()第一个参数为 自定义的事件名称第二个参数为需要传递的数据
* 父组件用v-on(@)绑定子组件定义的自定义事件名，监听子组件的事件，实现通信



### 1.4.3、事件总线

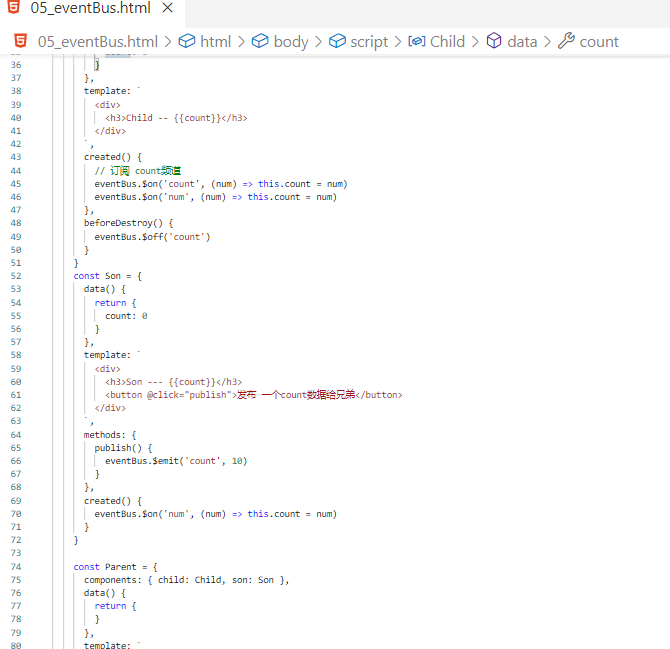
非父子组件或更多层级间组件间传值，在Vue中通过单独的事件中心来管理组件间的传值。



* 建立统一的事件中心

const bus = new Vue()

* 传递数据方，通过一个事件触发bus.$emit(方法名，传递的数据)
* 接收数据方，在生命周期函数中，通过bus.$on(方法名,[params])来监听
* 销毁事件，在接受数据方，通过bus.$off(方法名)销毁之后无法监听数据



### 1.4.4、Ref

ref被用来给元素或子组件注册引用信息。引用信息将会注册在父组件的 $refs 对象上。如果在普通的 DOM 元素上使用，引用指向的就是 DOM 元素；如果用在子组件上，引用就指向组件实例

# 普通dom元素上

<p ref="p">hello</p>

# 子组件上

<child-comp ref="child"></child-comp>

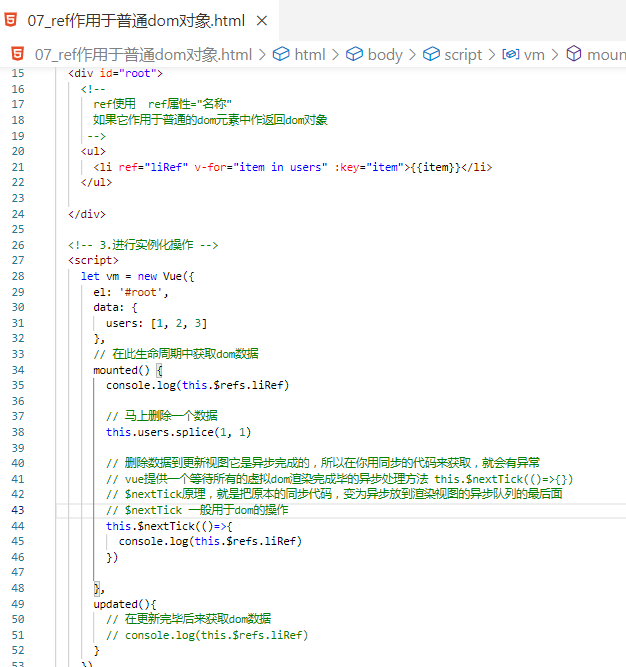
# 调用

this.$refs.引用名

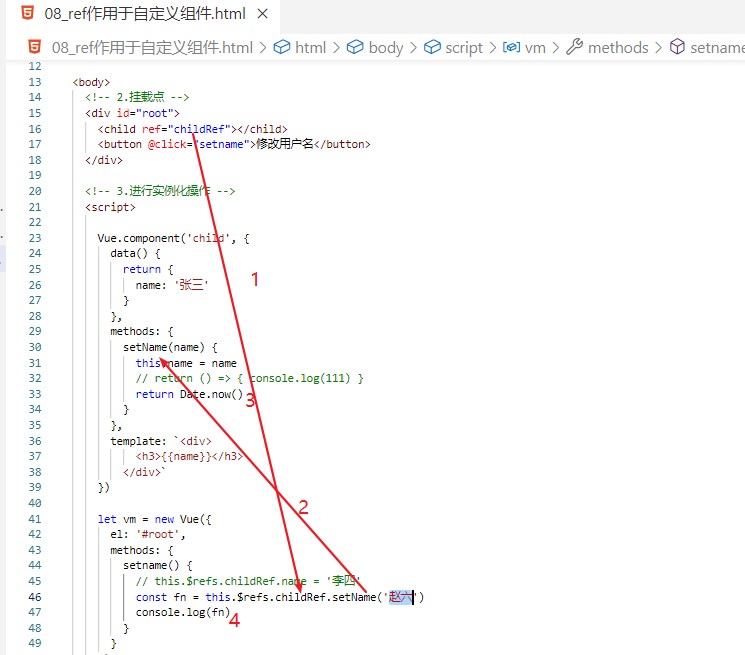
作用于普通的dom时，获取dom对象



获取最新的异步dom对象

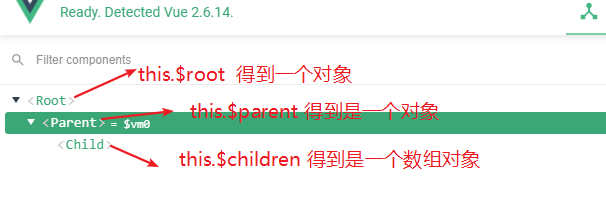


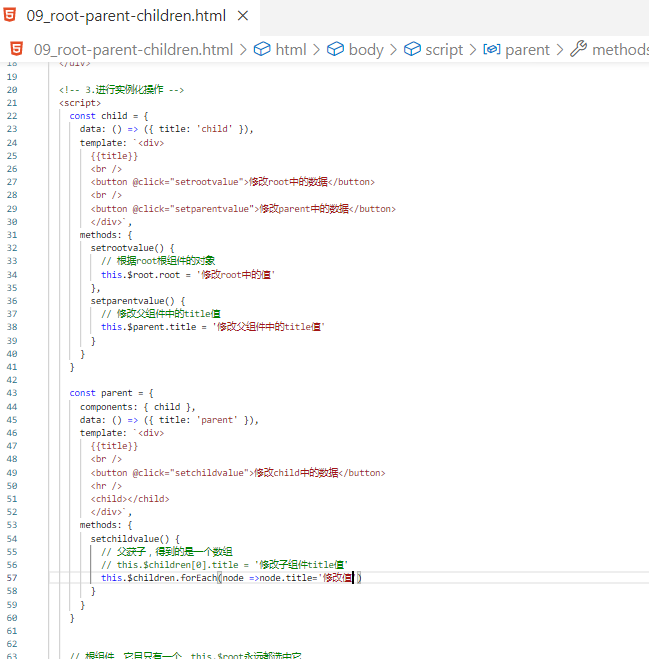
作用于自定义组件中，可以得到当前组件的实例对象，通过实例对象完成组件传值



### 1.5.5、$root/$parent/$children

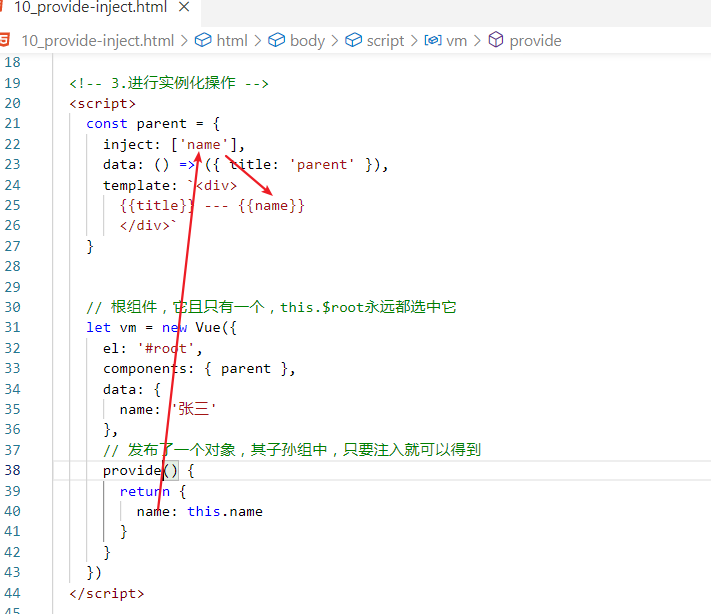
获取父组件对象或子组件对象集合





### 1.5.6、provide/inject

发布与注入，一般用于组件封装中使用



### 1.5.7、$attrs/$listeners

## 1.5、动态组件

通过使用保留的 <component> 元素，动态地绑定到它的 is 特性，我们让多个组件可以使用同一个挂载点，并动态切换

<component v-bind:is="currentView"></component>

<script>

var vm = new Vue({

el: '#root',

data: {

currentView: 'home'

},

components: {

home,

posts

}

})

</script>

* keep-alive

如果把切换出去的组件保留在内存中，可以保留它的状态或避免重新渲染。

<keep-alive>

<component :is="currentView"></component>

</keep-alive>

<script>

var vm = new Vue({

el: '#root',

data: {

currentView: 'posts'

},

components: {

home,

posts

}

})

</script>

<!-- 子组件中 -->

<script>

export default {

activated: function () {

console.log("home组件被激活了");

},

deactivated: function () {

console.log("home组件被停用了");

},

};

</script>

# 二、组件插槽

组件的最大特性就是复用性，而用好插槽能大大提高组件的可复用能力

## 2.1、单个插槽

当子组件模板只有一个没有属性的插槽时，父组件传入的整个内容片段将插入到插槽所在的 DOM 位置，并替换掉插槽标签本身。

# 子组件

<div>

<h2>我是子组件的标题</h2>

<slot>

只有在没有要分发的内容时才会显示。

</slot>

</div>

# 父组件

<div>

<h1>我是父组件的标题</h1>

<my-component>

<p>这是一些初始内容</p>

<p>这是更多的初始内容</p>

</my-component>

</div>

## 2.2、具名插槽

有时我们需要多个插槽，来完成对应的数据自定义显示。

一个不带 name 的 <slot> 出口会带有隐含的名字“default”。

自2.6.0 起有所更新。已废弃的使用 slot

# 子组件

<div class="container">

<header>

<slot name="header"></slot>

</header>

<main>

<slot></slot>

</main>

</div>

# 父组件

<app-layout>

// 老写法

<h1 slot="header">这里可能是一个页面标题</h1>

// 新写法

// v-slot 只能添加在 <template> 上

// 简写 v-slot:header == #header

<template v-slot:header>

<h1>这里可能是一个页面标题</h1>

</template>

<p>主要内容的一个段落。</p>

</app-layout>

## 2.3、作用域插槽

作用域插槽是一种特殊类型的插槽，用作一个 (能被传递数据的)可重用模板，来代替已经渲染好的元素。在子组件中，只需将数据传递到插槽，就像你将 prop 传递给组件一样

# 子组件中

Vue.component('child', {

template: `

<div class="child">

<slot name="default" text="我是子组件中的内容"></slot>

</div>

`

})

# 父组件中

<div class="parent">

<child>

// 老写法

<div name="default" slot-scope="props">

<div>父组件</div>

<h3>{{ props.text }}</h3>

</div>

// 新写法

<template #default="props"

<div>

<div>父组件</div>

<h3>{{ props.text }}</h3>

</div>

</child>

</div>

# 三、vue工程化

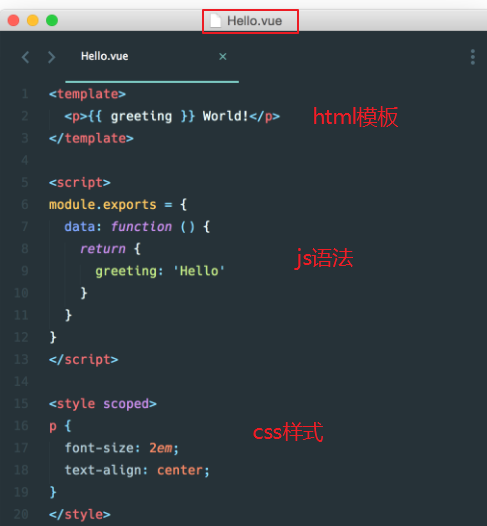
## 3.1、单文件组件

<https://cli.vuejs.org/zh/>

传统的网页引入vue.js使用有如下的问题

* 所有的组件都放同一个html文件中
* 没有构建步骤,不能使用npm来管理项目
* 缺乏语法高亮和提示
* 没有针对单个组件的css样式支持

针对于上述的问题，vue框架发布了vue-cli项目生成工具，Vue-cli是一个基于 Vue.js 进行快速开发的完整系统， 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。



vue-cli 3.x 它还没有在工程化中集成 vue3.0安装选项 dart-sass / node-sass

vue-cli 4.3之后 在vue-cli中就集成了 vue3.安装选项

## 3.2、工具安装

Vue-cli3.x以上版本需要 Node.js 8.9 或更高版本(推荐 v10 以上)。

npm i -g @vue/cli

## 安装成功后，检查

vue --version

## 3.3、创建项目

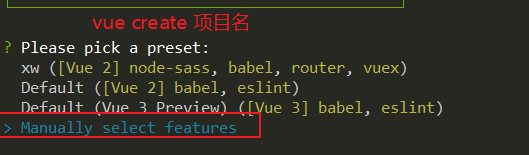
# 首先需要进入到对应的目录中(英文目录不要有空格),执行如下命令

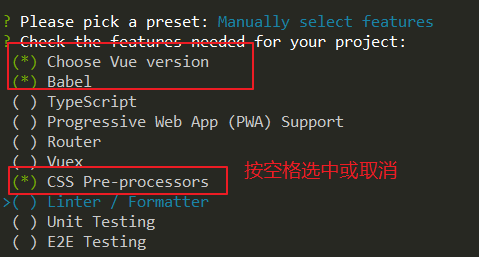
vue create 项目名称(创建时会自己以对应的项目名称生成目录)

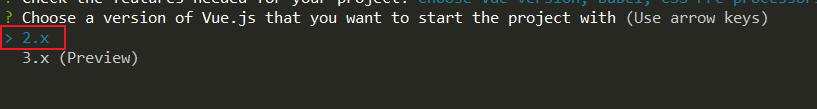
## 例

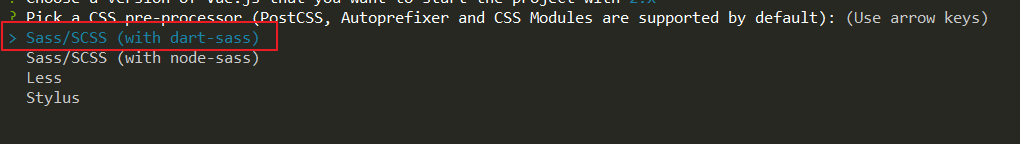
vue create hello-world

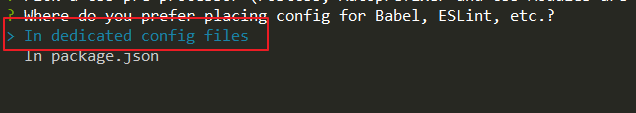
创建步骤



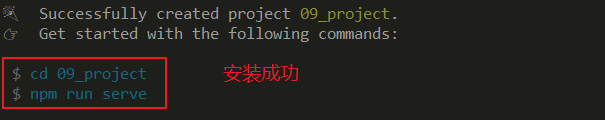












## 3.4、重要目录

public

index.html



src

main.js 项目入口文件

App.vue 根组件

components 自定义的功能组件

views 展示视图组件

assets 静态资源目录

