# 组件切换与keep-alive

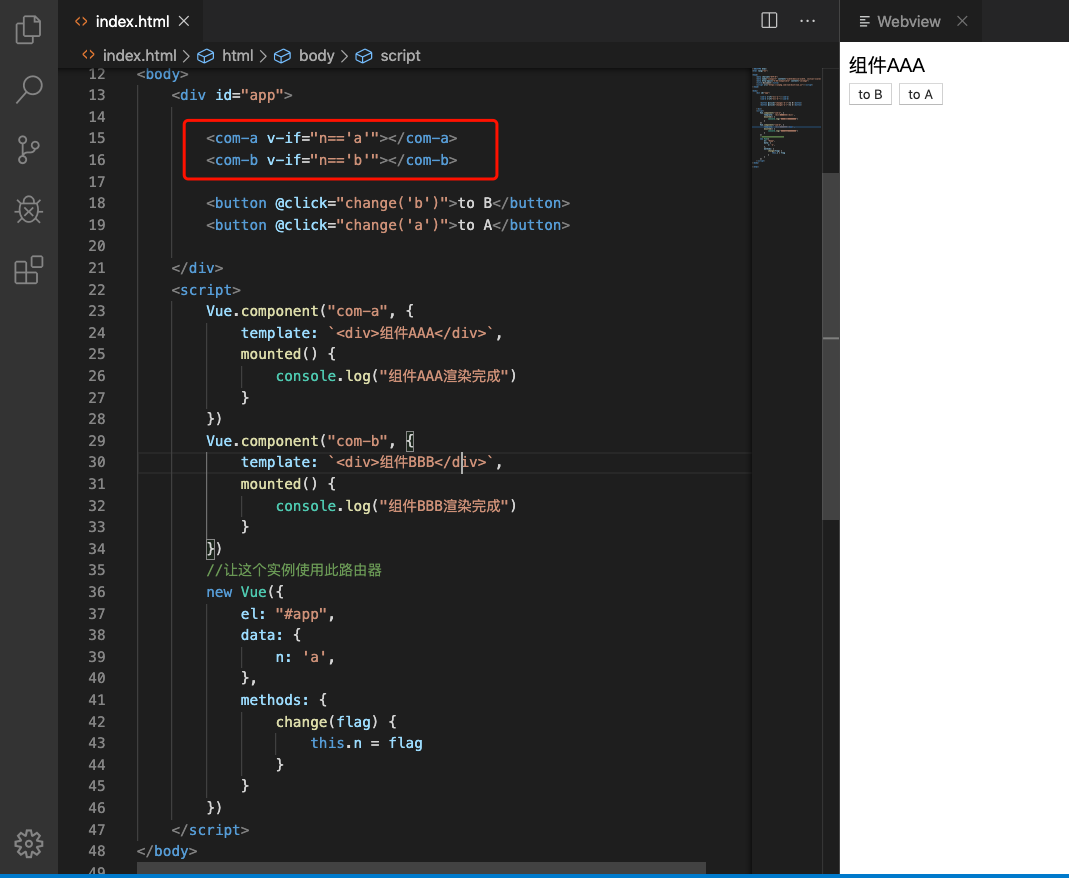
## v-if/动态组件/路由等，实现组件切换实行的是生成和创建操作

我们知道在vue中：动态组件（component+is）, v-if

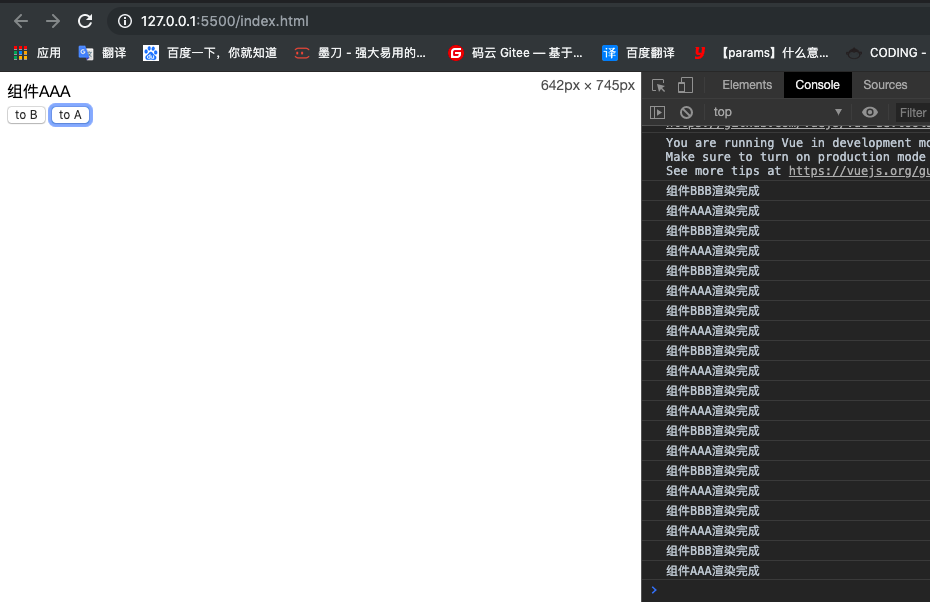
router-view等导致的组件切换，实现的是销毁组件和重新生组件的操作。

**下面是一些例子：**

### V-if实现组件切换



在这个例子中，我们通过了v-if指令实现了组件的切换，这时会发现每次切换，渲染完成的钩子函数都会执行，说明每次组件都是重新创建的。



<!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>

<script src="https://unpkg.com/vue/dist/vue.js"></script>

</head>

<body>

<div id="app">

<com-a v-if="n=='a'"></com-a>

<com-b v-if="n=='b'"></com-b>

<button @click="change('b')">to B</button>

<button @click="change('a')">to A</button>

</div>

<script>

Vue.component("com-a", {

template: `<div>组件AAA</div>`,

mounted() {

console.log("组件AAA渲染完成")

}

})

Vue.component("com-b", {

template: `<div>组件BBB</div>`,

mounted() {

console.log("组件BBB渲染完成")

}

})

//让这个实例使用此路由器

new Vue({

el: "#app",

data: {

n: 'a',

},

methods: {

change(flag) {

this.n = flag

}

}

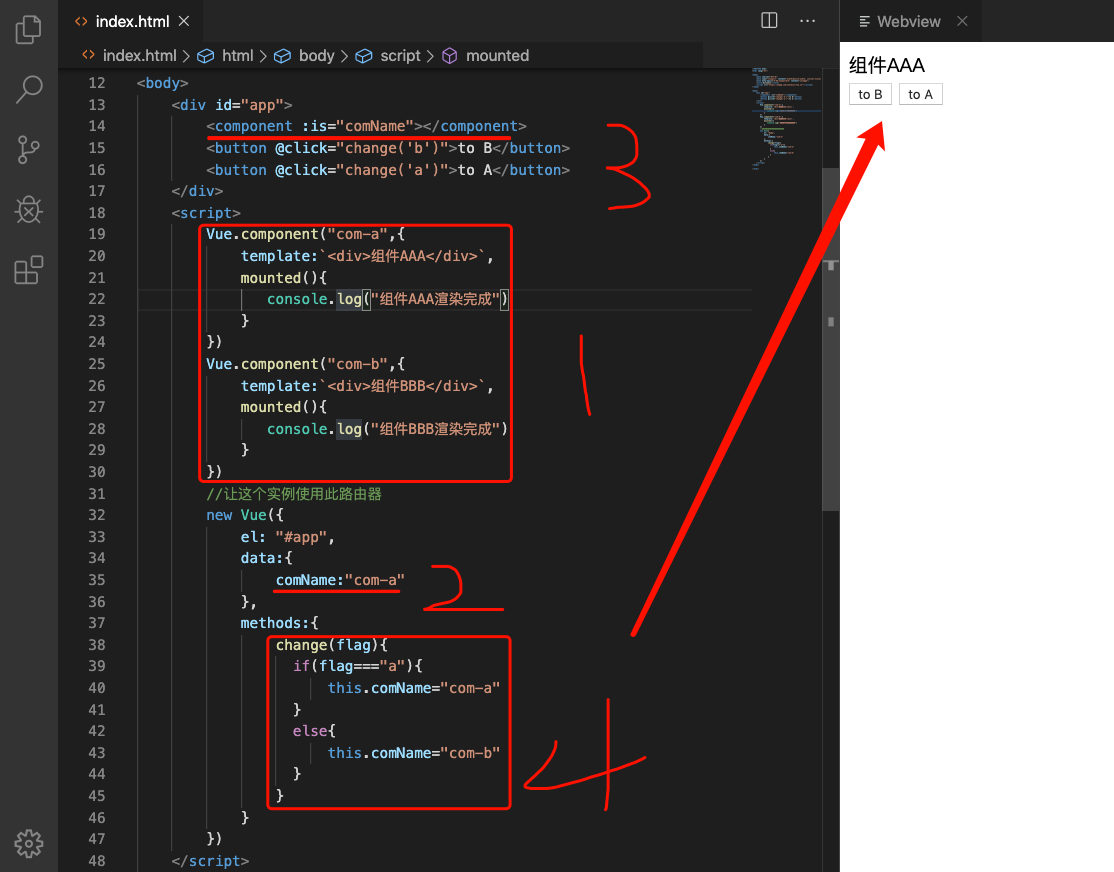
})

</script>

</body>

</html>

### **component**+is动态组件



在这个例子中，我们通过了component+is实现了组件的切换，这时会发现每次切换，渲染完成的钩子函数都会执行，说明每次组件都是重新创建的。



完整代码

<!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>

<script src="https://unpkg.com/vue/dist/vue.js"></script>

</head>

<body>

<div id="app">

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

<button @click="change('b')">to B</button>

<button @click="change('a')">to A</button>

</div>

<script>

Vue.component("com-a",{

template:`<div>组件AAA</div>`,

mounted(){

console.log("组件AAA渲染完成")

}

})

Vue.component("com-b",{

template:`<div>组件BBB</div>`,

mounted(){

console.log("组件BBB渲染完成")

}

})

//让这个实例使用此路由器

new Vue({

el: "#app",

data:{

comName:"com-a"

},

methods:{

change(flag){

if(flag==="a"){

this.comName="com-a"

}

else{

this.comName="com-b"

}

}

}

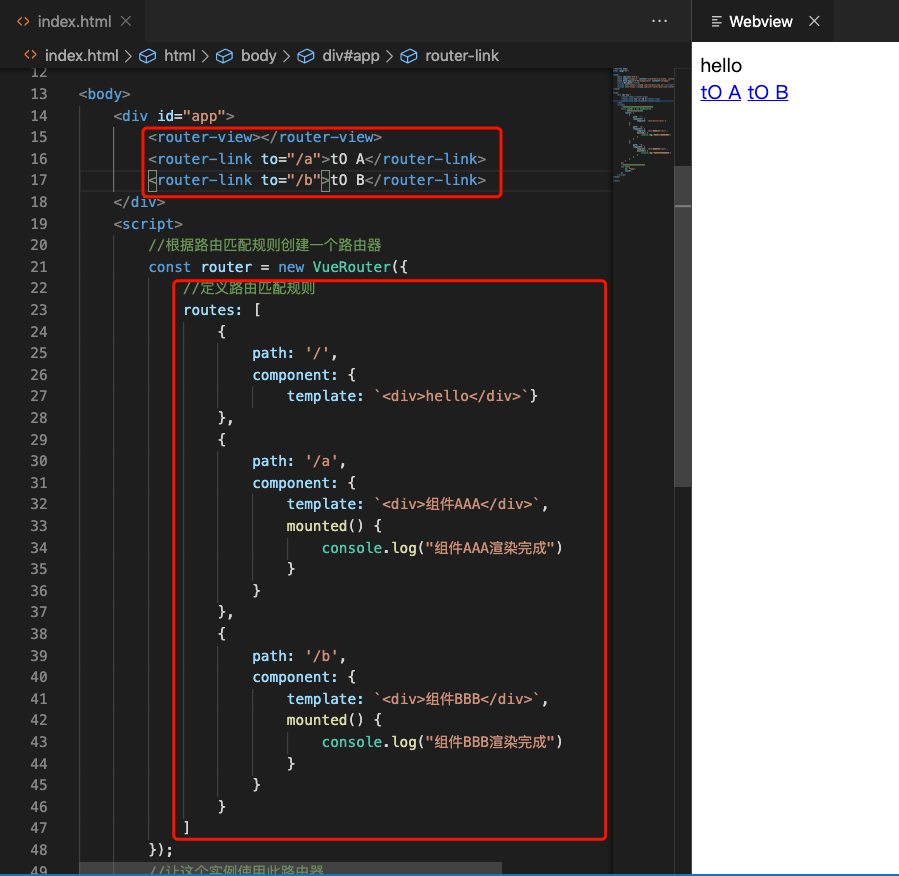
})

</script>

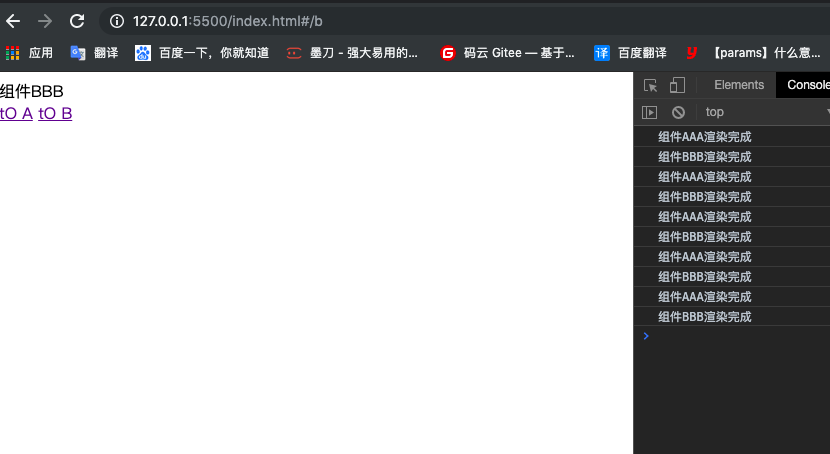
</body>

</html>

### 路由router实现组件切换



在这个例子中，我们通过了路由实现了组件的切换，这时会发现每次切换，渲染完成的钩子函数都会执行，说明每次组件都是重新创建的。



完整代码

<!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>

<script src="https://unpkg.com/vue/dist/vue.js"></script>

<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>

</head>

<body>

<div id="app">

<router-view></router-view>

<router-link to="/a">tO A</router-link>

<router-link to="/b">tO B</router-link>

</div>

<script>

//根据路由匹配规则创建一个路由器

const router = new VueRouter({

//定义路由匹配规则

routes: [

{

path: '/',

component: {

template: `<div>hello</div>`}

},

{

path: '/a',

component: {

template: `<div>组件AAA</div>`,

mounted() {

console.log("组件AAA渲染完成")

}

}

},

{

path: '/b',

component: {

template: `<div>组件BBB</div>`,

mounted() {

console.log("组件BBB渲染完成")

}

}

}

]

});

//让这个实例使用此路由器

new Vue({

el: "#app",

router,

})

</script>

</body>

</html>

## keep-alive---会缓存不活动的组件实例，而不是销毁它们

### V-if

对于v-if而言，在条件外面包裹一个keep-alive标签即能实现：缓存不活动的组件实例，而不是销毁它们

<keep-alive>

<com-a v-if="n=='a'"></com-a>

<com-b v-if="n=='b'"></com-b>

</keep-alive>

### 动态组件

对于动态组件而言，在component外面包裹一个keep-alive标签即能实现：缓存不活动的组件实例，而不是销毁它们

<keep-alive>

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

</keep-alive>

### 路由

对于路由跳转而言，在router-view外面包裹一个keep-alive标签即能实现：缓存不活动的组件实例，而不是销毁它们

<keep-alive>

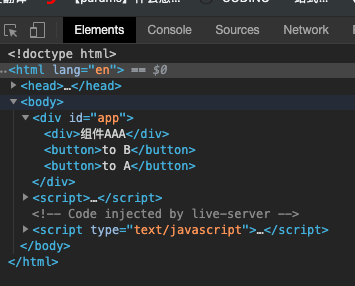
<router-view></router-view>

</keep-alive>

## **keep-alive---详细说明**

### <keep-alive> 本身不会渲染

<keep-alive> 是一个抽象组件：类似于template标签，它自身不会作为一个 DOM 元素显然，只会渲染里面包裹的内容。



### <keep-alive> 中同一时刻只能有一个组件被渲染

当在keep-alive中操作v-if时，如果有多个条件性的组件，<keep-alive> 要求同时只有一个组件被渲染。

如在下面的代码中，当n=b时，只会渲染和缓存com-b组件，不会渲染和缓存com-c组件

<keep-alive>

<com-a v-if="n=='a'"></com-a>

<com-b v-if="n=='b'"></com-b>

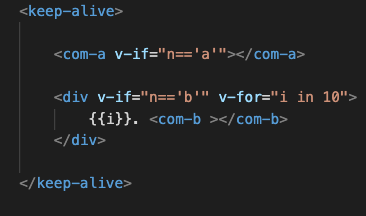
<com-c v-if="n=='b'"></com-c>

</keep-alive>

### <keep-alive>中不能通过v-for循环创建组件

由于一个时刻，keep-alive只能渲染和缓存一个组件，所以如果在keep-alive中有 v-for ，则keep-alive不会工作

在下面的代码里面，com-b组件由于在for循环里面，所以就不会渲染



<keep-alive>

<com-a v-if="n=='a'"></com-a>

<div v-if="n=='b'" v-for="i in 10">

{{i}}. <com-b ></com-b>

</div>

</keep-alive>

### ke[ep-alive](https://cn.vuejs.org/v2/api/" \l "keep-alive" \o "keep-alive)有三个参数

include - 字符串或正则表达式。只有名称匹配的组件会被缓存。

exclude - 字符串或正则表达式。任何名称匹配的组件都不会被缓存。

max - 数字。最多可以缓存多少组件实例。

#### include 和 exclude 属性允许组件有条件地缓存

二者都可以用逗号分隔字符串、正则表达式或一个数组来表示：



匹配首先检查组件自身的 name 选项，如果 name 选项不可用，则匹配它的局部注册名称 (父组件 components 选项的键值)。匿名组件不能被匹配。

#### max表示最多可以缓存的组件实例个数

最多可以缓存多少组件实例。一旦这个数字达到了，在新实例被创建之前，已缓存组件中最久没有被访问的实例会被销毁掉。

<keep-alive :max="10">

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

</keep-alive>

<keep-alive> 不会在函数式组件中正常工作，因为它们没有缓存实例。

# **组件的另外三个钩子函数**

[activated](https://cn.vuejs.org/v2/api/" \l "activated) keep-alive 组件激活时调用。

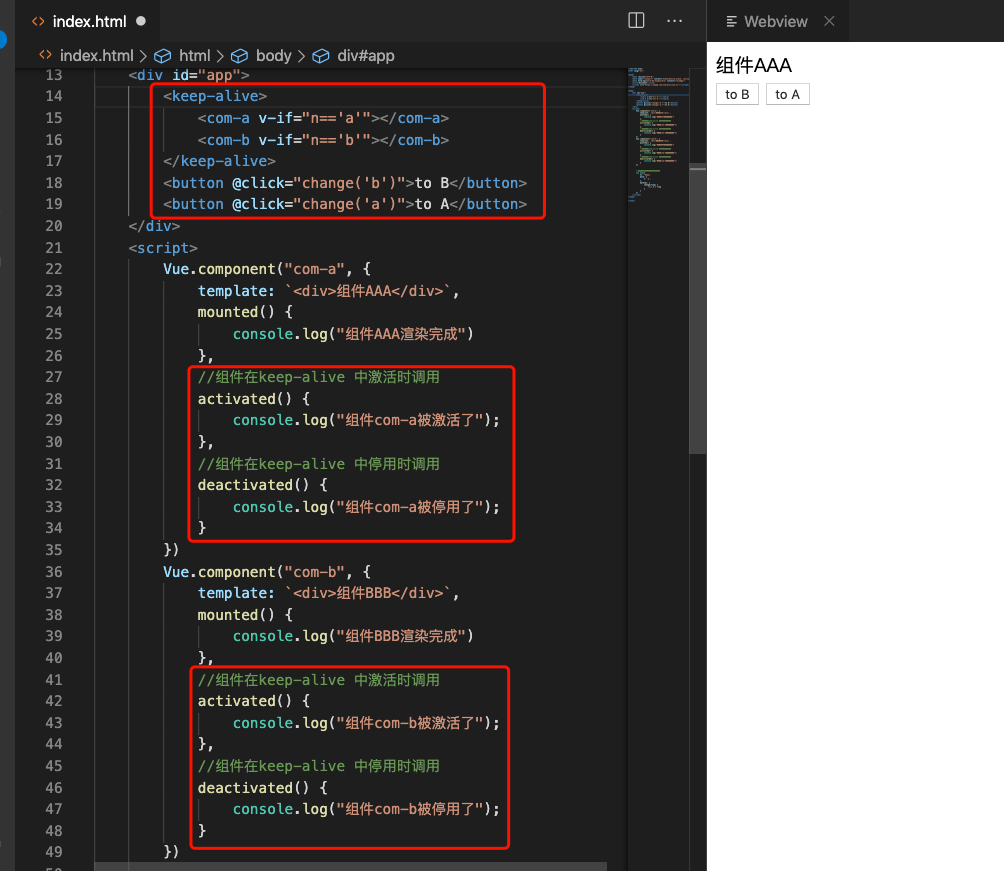
[deactivated](https://cn.vuejs.org/v2/api/" \l "deactivated) keep-alive 组件停用时调用

[errorCaptured](https://cn.vuejs.org/v2/api/" \l "errorCaptured" \o "errorCaptured) 当捕获一个来自子孙组件的错误时被调用

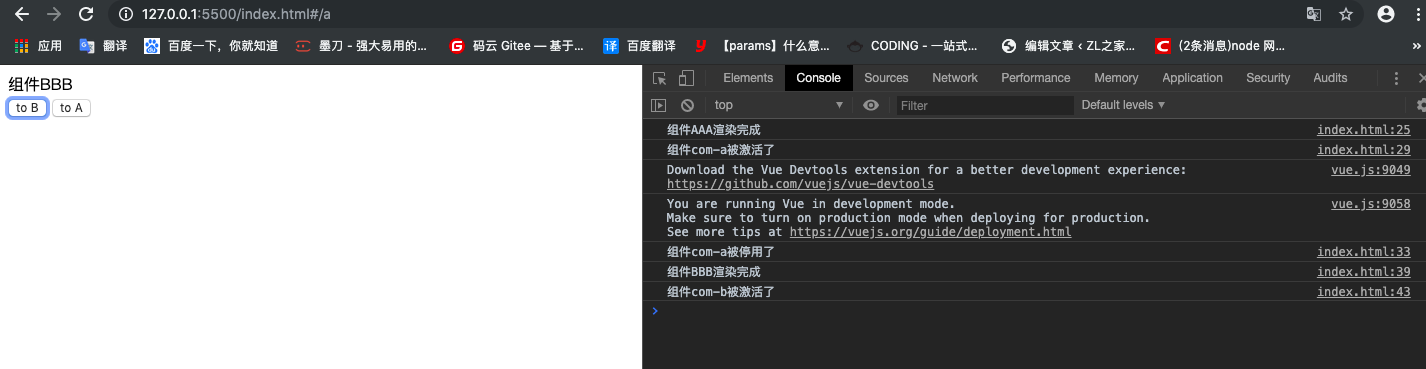
## **[activated](https://cn.vuejs.org/v2/api/" \l "activated)与[deactivated](https://cn.vuejs.org/v2/api/" \l "deactivated)钩子函数---**主要受keep-alive影响

当组件在 <keep-alive> 内被切换，它的 activated 和 deactivated 这两个生命周期钩子函数将会被对应执行。

1. [activated](https://cn.vuejs.org/v2/api/" \l "activated) 组件在keep-alive 中激活时调用。
2. [deactivated](https://cn.vuejs.org/v2/api/" \l "deactivated) 组件在keep-alive 中停用时调用



效果演示



完整代码

<!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>

<script src="https://unpkg.com/vue/dist/vue.js"></script>

</head>

<body>

<div id="app">

<keep-alive>

<com-a v-if="n=='a'"></com-a>

<com-b v-if="n=='b'"></com-b>

</keep-alive>

<button @click="change('b')">to B</button>

<button @click="change('a')">to A</button>

</div>

<script>

Vue.component("com-a", {

template: `<div>组件AAA</div>`,

mounted() {

console.log("组件AAA渲染完成")

},

//组件在keep-alive 中激活时调用

activated() {

console.log("组件com-a被激活了");

},

//组件在keep-alive 中停用时调用

deactivated() {

console.log("组件com-a被停用了");

}

})

Vue.component("com-b", {

template: `<div>组件BBB</div>`,

mounted() {

console.log("组件BBB渲染完成")

},

//组件在keep-alive 中激活时调用

activated() {

console.log("组件com-b被激活了");

},

//组件在keep-alive 中停用时调用

deactivated() {

console.log("组件com-b被停用了");

}

})

//让这个实例使用此路由器

new Vue({

el: "#app",

data: {

n: 'a',

},

methods: {

change(flag) {

this.n = flag

}

}

})

</script>

</body>

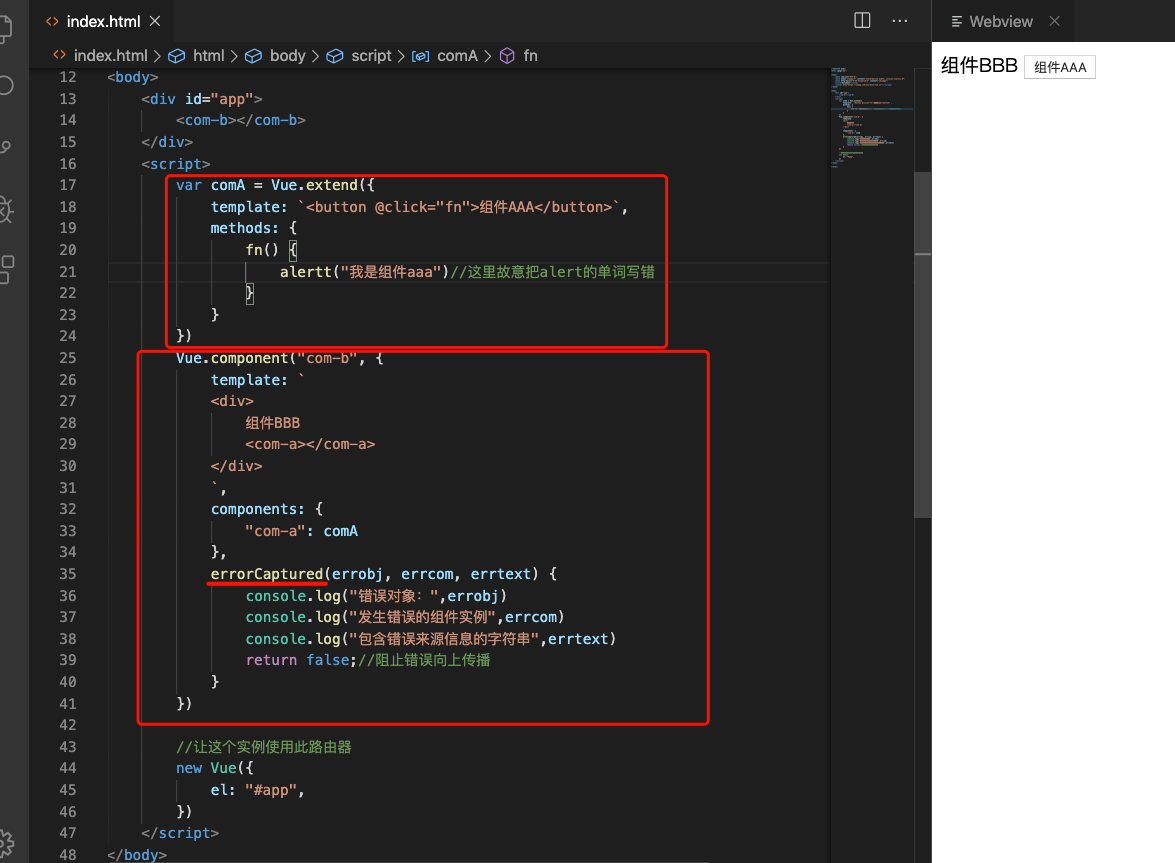
</html>

参考地址：[https://cn.vuejs.org/v2/api/#activated](https://cn.vuejs.org/v2/api/" \l "activated)

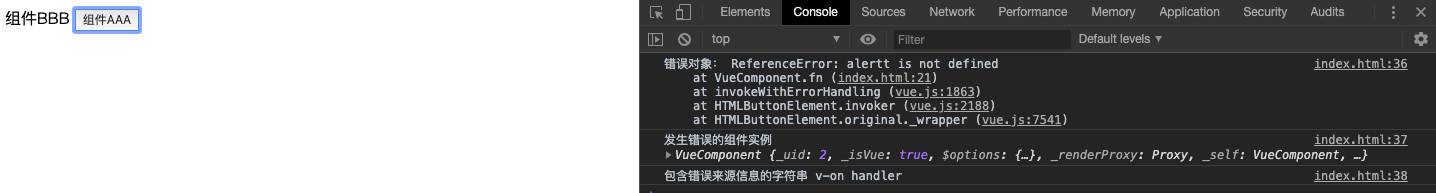
## **[errorCaptured](https://cn.vuejs.org/v2/api/" \l "errorCaptured" \o "errorCaptured) 当捕获一个来自子孙组件的错误时被调用**

### 说明

此钩子会收到三个参数：错误对象、发生错误的组件实例以及一个包含错误来源信息的字符串。此钩子可以返回 false 以阻止该错误继续向上传播。



### 效果



### 完整代码

<!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>

<script src="https://unpkg.com/vue/dist/vue.js"></script>

</head>

<body>

<div id="app">

<com-b></com-b>

</div>

<script>

var comA = Vue.extend({

template: `<button @click="fn">组件AAA</button>`,

methods: {

fn() {

alertt("我是组件aaa")//这里故意把alert的单词写错

}

}

})

Vue.component("com-b", {

template: `

<div>

组件BBB

<com-a></com-a>

</div>

`,

components: {

"com-a": comA

},

errorCaptured(errobj, errcom, errtext) {

console.log("错误对象：",errobj)

console.log("发生错误的组件实例",errcom)

console.log("包含错误来源信息的字符串",errtext)

return false;//阻止错误向上传播

}

})

//让这个实例使用此路由器

new Vue({

el: "#app",

})

</script>

</body>

</html>

### 全局的 config.errorHandler



### 错误传播规则

1. 如果一个组件的继承或父级从属链路中存在多个 errorCaptured 钩子，则它们将会被相同的错误逐个唤起。
2. 默认情况下，如果全局的 config.errorHandler 被定义，所有的错误仍会发送它，因此这些错误仍然会向单一的分析服务的地方进行汇报。
3. 如果此 errorCaptured 钩子自身抛出了一个错误，则这个新错误和原本被捕获的错误都会发送给全局的 config.errorHandler。
4. 一个 errorCaptured 钩子能够返回 false 以阻止错误继续向上传播。本质上是说“这个错误已经被搞定了且应该被忽略”。它会阻止其它任何会被这个错误唤起的 errorCaptured 钩子和全局的 config.errorHandler。

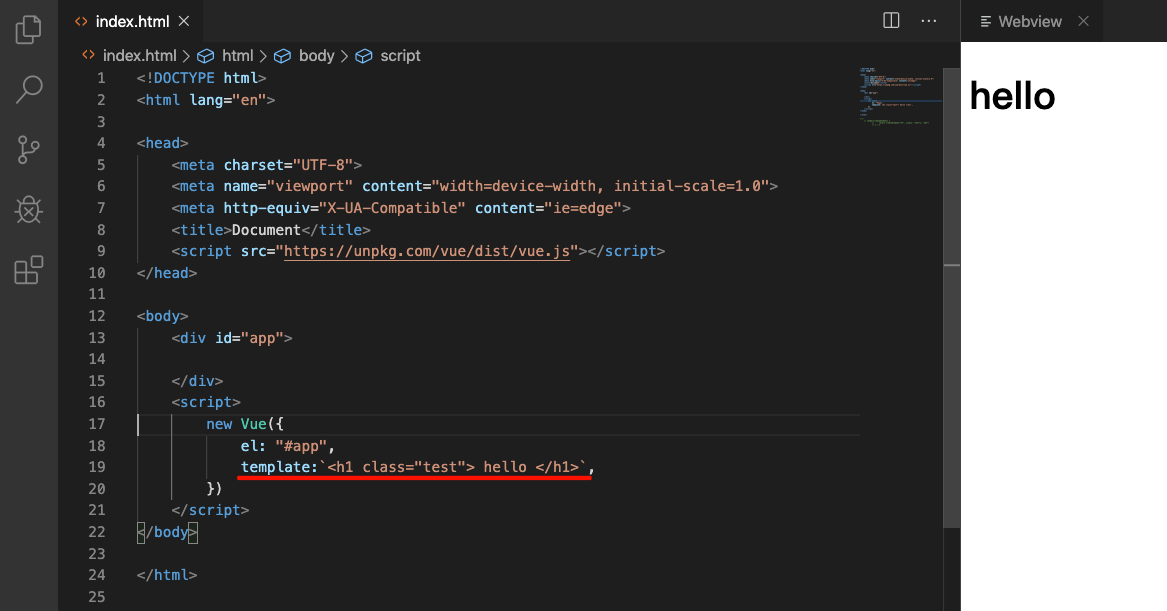
# 使用render方式渲染组件

## 说明

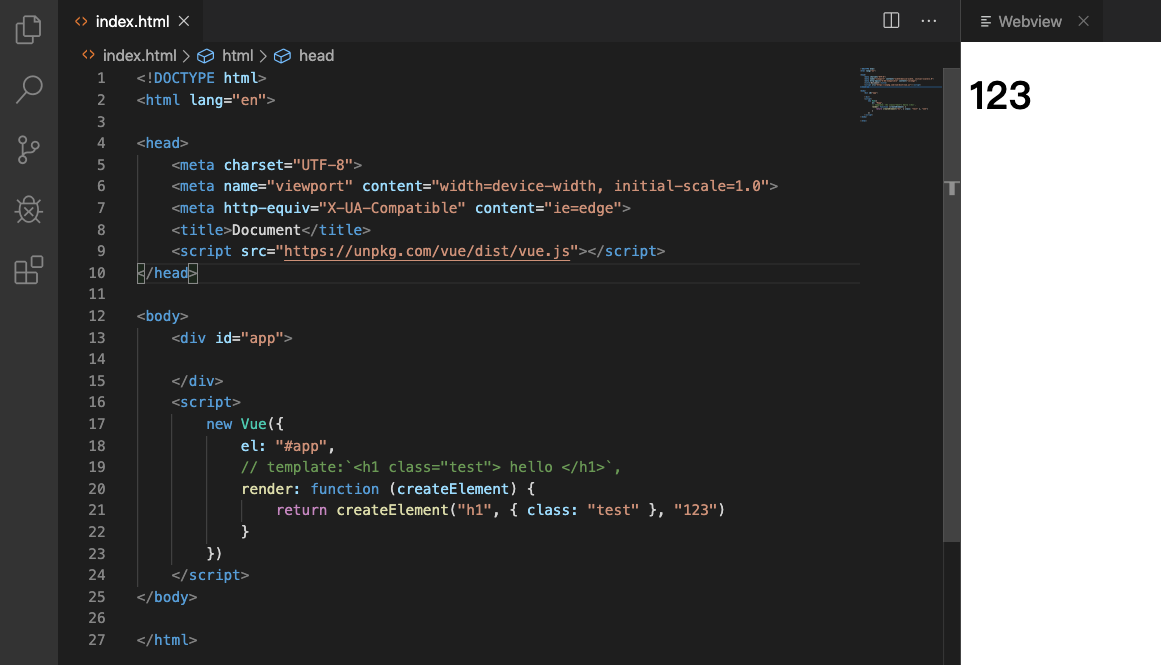
Vue 推荐在绝大多数情况下使用 template 来创建你的 HTML模板。然而在一些场景中，你真的需要 JavaScript 的完全编程的能力，这时你可以用 ****render 函数****，它比 template 更接近编译器。

实际上: Vue 的模板实际是编译成了 render 函数。

### 使用template模板渲染



### **使用render函数来渲染**



## **[createElement](https://cn.vuejs.org/v2/guide/render-function.html" \l "createElement-%E5%8F%82%E6%95%B0" \o "createElement 参数)**

### 基本语法

接下来你需要熟悉的是如何在 createElement 函数中生成模板。

这里是 createElement 接受的参数：

第一个参数可以是HTML标签名，组件或者函数都可以；此参数是必须的；

第二个参数：代表了标签上各属性的数据集合，对象类型（可选）；

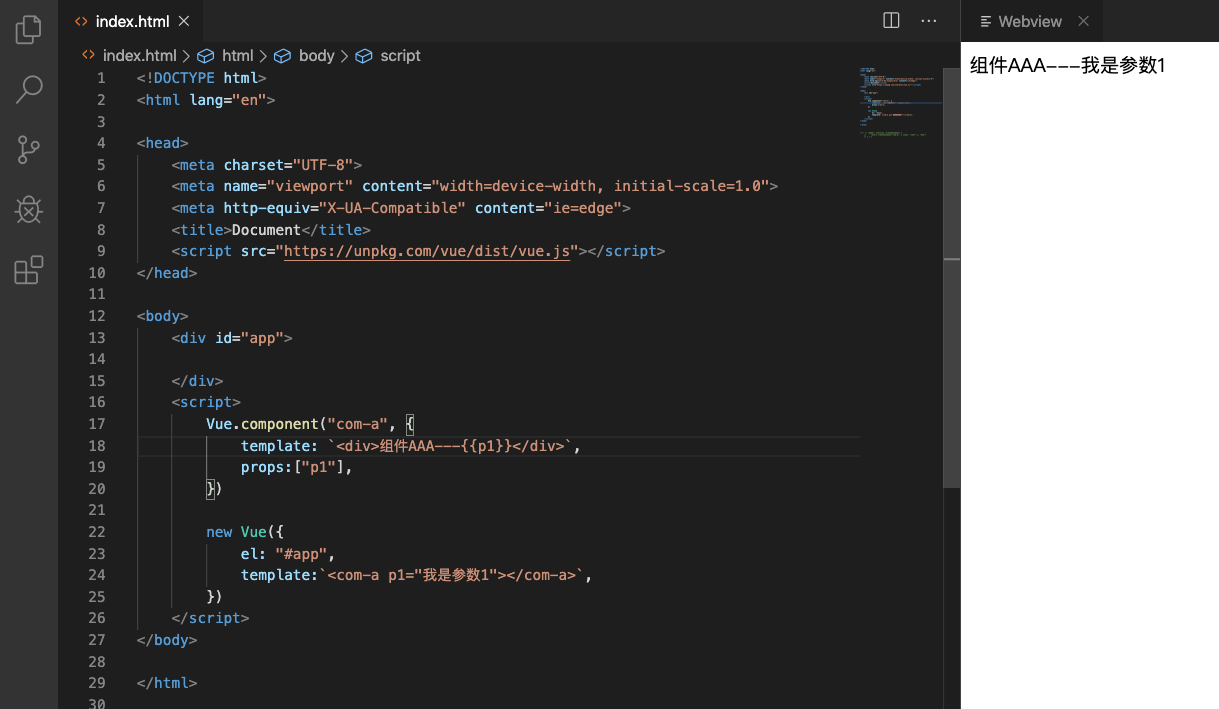
第三个参数：代表标签内容，字符串或数组类型（可选）。

我们可以通过createElement来创建我们想要的template节点

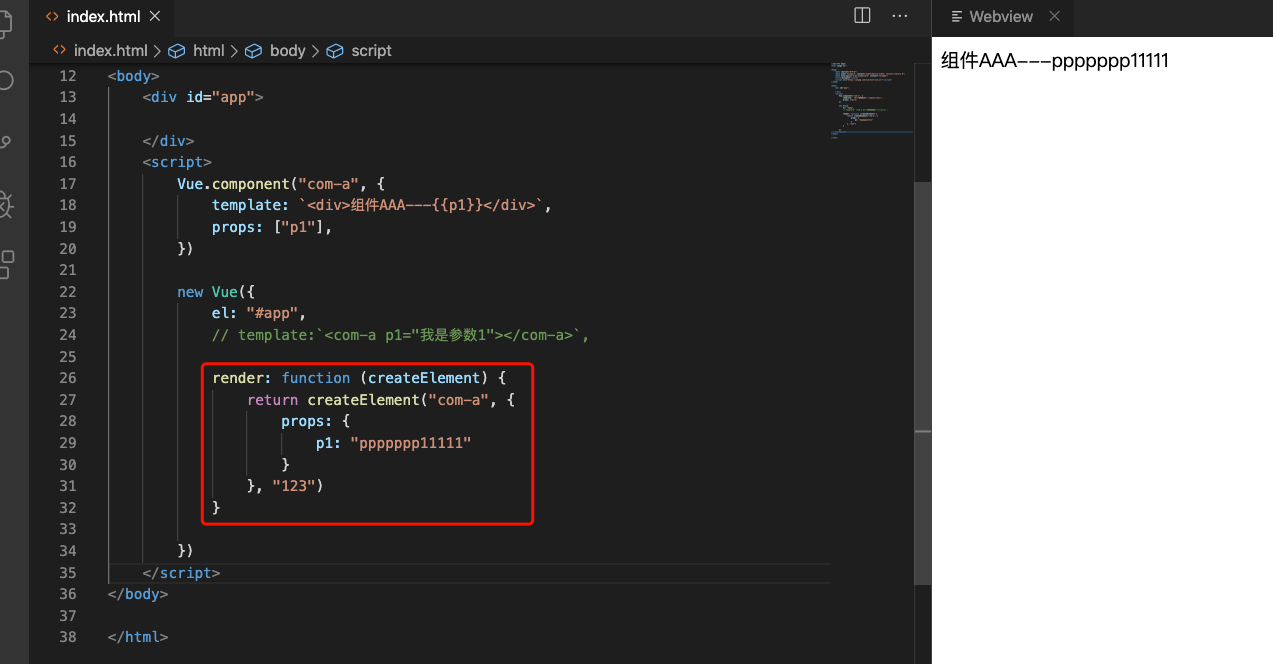
### 具体使用

#### 使用createElement直接渲染一个组件

使用tempate时



使用createElement时



代码

<!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>

<script src="https://unpkg.com/vue/dist/vue.js"></script>

</head>

<body>

<div id="app">

</div>

<script>

Vue.component("com-a", {

template: `<div>组件AAA---{{p1}}</div>`,

props: ["p1"],

})

new Vue({

el: "#app",

// template:`<com-a p1="我是参数1"></com-a>`,

render: function (createElement) {

return createElement("com-a", {

props: {

p1: "ppppppp11111"

}

}, "123")

}

})

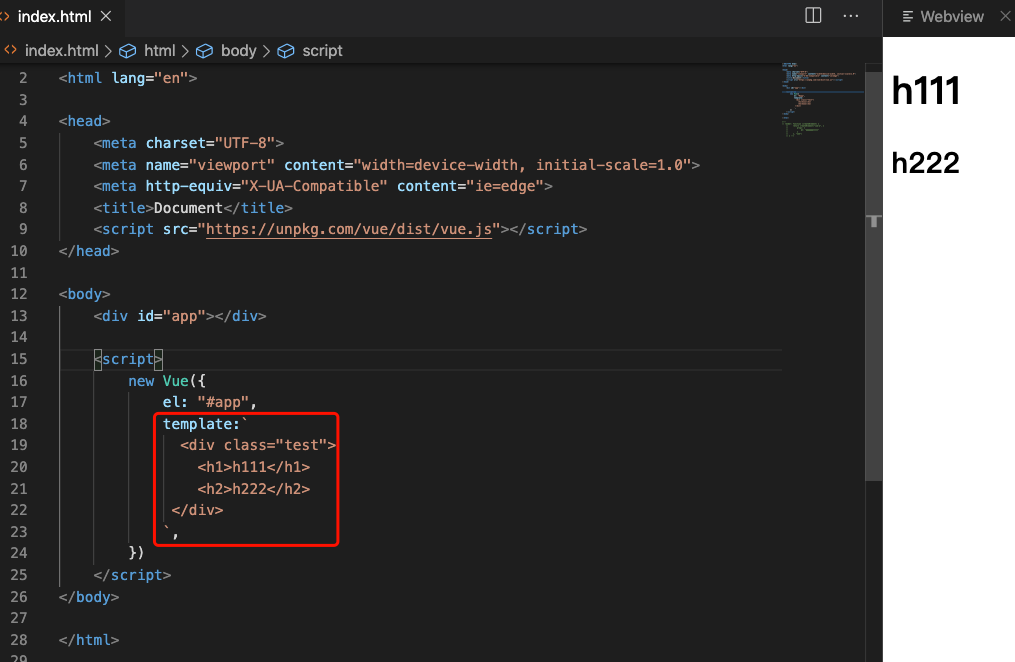
</script>

</body>

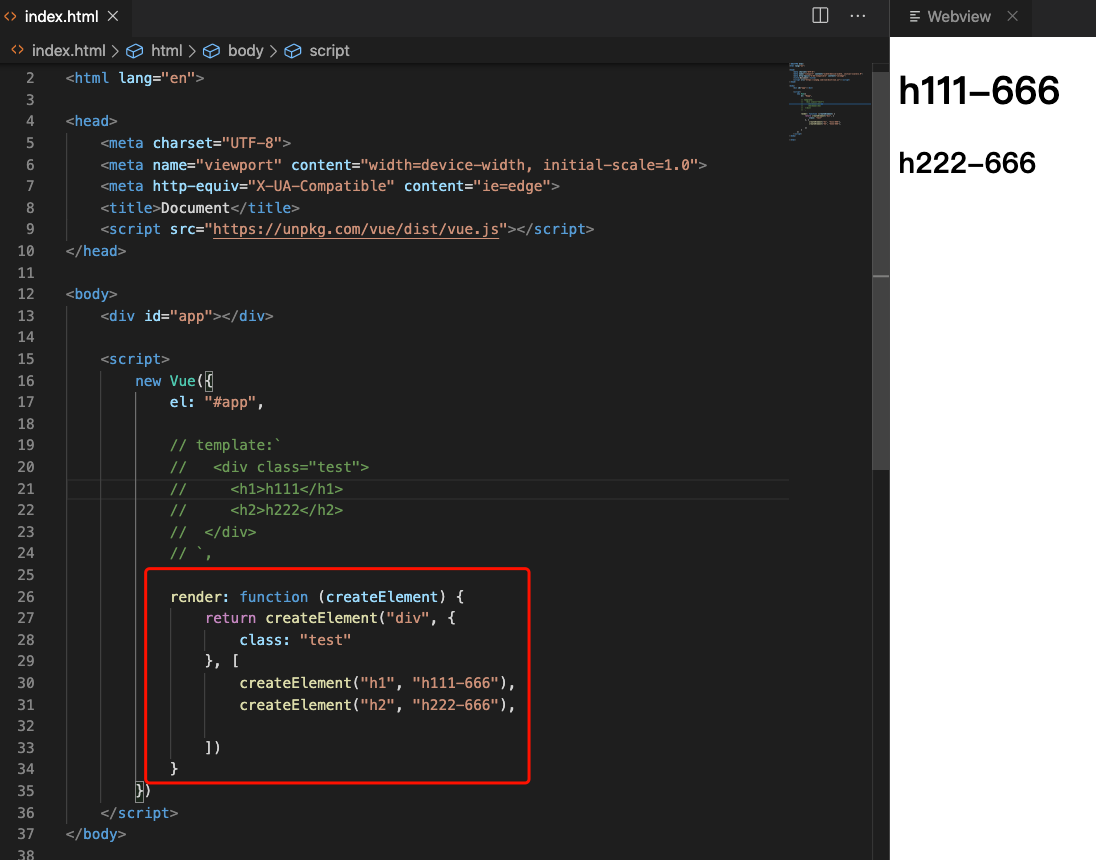
</html>

#### 使用createElement直接渲染一个包含子标签的div

使用template时



使用createElement时



代码

<!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>

<script src="https://unpkg.com/vue/dist/vue.js"></script>

</head>

<body>

<div id="app"></div>

<script>

new Vue({

el: "#app",

// template:`

// <div class="test">

// <h1>h111</h1>

// <h2>h222</h2>

// </div>

// `,

render: function (createElement) {

return createElement("div", {

class: "test"

}, [

createElement("h1", "h111-666"),

createElement("h2", "h222-666"),

])

}

})

</script>

</body>

</html>

### 代表标签属性的数据集合对象

如果你还想通过crerateElement创建模板时绑定各种事件，如点击事件等或其他操作，那么你就需要了解这个代表标签属性的数据集合对象

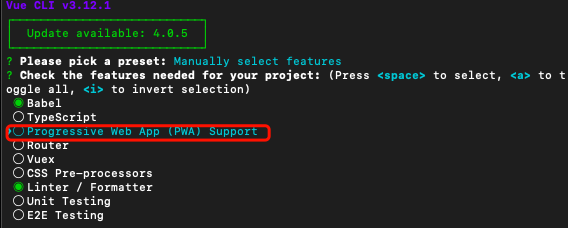
|  |
| --- |
| {  // 和`v-bind:class`一样的 API  // 接收一个字符串、对象或字符串和对象组成的数组  'class': {  foo: true,  bar: false  },  // 和`v-bind:style`一样的 API  // 接收一个字符串、对象或对象组成的数组  style: {  color: 'red',  fontSize: '14px'  },  // 普通的 HTML 特性  attrs: {  id: 'foo'  },  // 组件 props  props: {  myProp: 'bar'  },  // DOM 属性  domProps: {  innerHTML: 'baz'  },  // 事件监听器基于 `on`  // 所以不再支持如 `v-on:keyup.enter` 修饰器  // 需要手动匹配 keyCode。  on: {  click: this.clickHandler  },   // 仅用于组件，用于监听原生事件，而不是组件内部使用  // `vm.$emit` 触发的事件。  nativeOn: {  click: this.nativeClickHandler  },   // 自定义指令。注意，你无法对 `binding` 中的 `oldValue`  // 赋值，因为 Vue 已经自动为你进行了同步。   directives: [  {  name: 'my-custom-directive',  value: '2',  expression: '1 + 1',  arg: 'foo',  modifiers: {  bar: true  }  }  ],   // 作用域插槽格式  // { name: props => VNode | Array<VNode> }  scopedSlots: {  default: props => createElement('span', props.text)  },   // 如果组件是其他组件的子组件，需为插槽指定名称   slot: 'name-of-slot',  // 其他特殊顶层属性  key: 'myKey',  ref: 'myRef',  // 如果你在渲染函数中向多个元素都应用了相同的 ref 名，  // 那么 `$refs.myRef` 会变成一个数组。  refInFor: true } |

## **更多的内容见官网文档**

[https://cn.vuejs.org/v2/guide/render-function.html#%E4%BD%BF%E7%94%A8-JavaScript-%E4%BB%A3%E6%9B%BF%E6%A8%A1%E6%9D%BF%E5%8A%9F%E8%83%BD](https://cn.vuejs.org/v2/guide/render-function.html" \l "%E4%BD%BF%E7%94%A8-JavaScript-%E4%BB%A3%E6%9B%BF%E6%A8%A1%E6%9D%BF%E5%8A%9F%E8%83%BD)

# 使用脚手架创建项目时的各个选项了解【可选】

## Progressive Web App (PWA) Support--渐进式web应用



眼下的移动端APP

* Native APP ----原生ios或者android
* Web App ----运行到手机浏览器里面的web网页
* Hybrid App----webview运行网页，从而在手机上运行
* PWA(Progressive Web App)---渐进式webapp

### ****Native APP****

Native APP，指原生App。是一个完整的App。可拓展性强，须要用户下载安装使用。

长处：

* 可使用移动设备全部功能
* 速度快、性能高、用户体验好
* 离线使用

缺点：

* 开发成本高、维护成本高
* 每一个不通的平台都要又一次开发
* 应用商店审核复杂，效率低

### ****Web APP****

Web App 指採用Html5语言写出的App。生活在浏览器里的应用。不须要下载安装。

长处：

* 跨平台开发、无需下载、无需安装,开发速度快
* 公布灵活。由于根本不须要应用商店的审核
* 较低的开发成本
* 可即时上线
* 用户能够直接使用最新版本号
* 支持设备广泛

缺点：

* 仅仅能使用有限的移动设备API
* 浏览器兼容问题
* 无法上传到应用商店
* 用户临时不适用

### ****Hybrid App****

Hybrid APP指的是半原生半Web的混合类App，须要下载安装。

长处：

* 兼容多平台
* Web前端工作人员就可高速构建
* 能够上传到应用商店
* 能够基于浏览器的方式进行页面调试
* 可使用的移动设备的API多

缺点：

* 用户体验不如原生应用
* 为模拟原生样式。须要大量的html和css
* 性能稍慢
* 技术不是非常成熟

### PWA

#### 什么是PWA

PWA全称Progressive Web App，直译是渐进式WEB应用，

是 Google 在 2015 年提出，2016年6月才推广的项目。

是结合了一系列现代Web技术的组合。在网页应用中实现和原生应用相近的用户体验。

所谓的P（Progressive）这里有两层含义，一方面是渐进增强，让WEB APP的体验和功能能够用渐进增强的方式来更接近原生APP的体验及功能。另一方面是指下一代WEB技术。PWA并非描写叙述一个技术。而是一些技术的合集。

#### PWA关键技术

##### ****Manifest（应用清单）****

Web App Manifest是一个W3C规范，定义了一个基于JSON的清单，为开发人员提供一个放置与Web应用程序关联的元数据的集中地点。manifest 就是 PWA 概念的一环，它给你了控制你的应用怎样出如今用户期待出现的地方（比方用户手机主屏幕）。这直接影响到用户能启动什么。以及更重要的。用户怎样启动它。

使用 web 应用程序清单，你的应用能够：

* 能够真实存在于用户主屏幕上
* 在 Android 上能够全屏启动，不显示地址栏
* 控制屏幕方向已获得最佳效果
* 定义启动画面，为你的站点定义主题
* 追踪你的应用是从主屏幕还是 URL 启动的

比如：

{

"lang": "en",

"dir": "ltr",

"name": "Donate App",

"description": "This app helps you donate to worthy causes.",

"short\_name": "Donate",

"icons": [{

"src": "icon/lowres.webp",

"sizes": "64x64",

"type": "image/webp"

},{

"src": "icon/lowres.png",

"sizes": "64x64"

}, {

"src": "icon/hd\_hi",

"sizes": "128x128"

}],

"scope": "/racer/",

"start\_url": "/racer/start.html",

"display": "fullscreen",

"orientation": "landscape",

"theme\_color": "aliceblue",

"background\_color": "red",

"serviceworker": {

"src": "sw.js",

"scope": "/racer/",

"use\_cache": false

},

"screenshots": [{

"src": "screenshots/in-game-1x.jpg",

"sizes": "640x480",

"type": "image/jpeg"

},{

"src": "screenshots/in-game-2x.jpg",

"sizes": "1280x920",

"type": "image/jpeg"

}]

}

#### PWA优、缺点

长处

· ****渐进增强**** - 能够让每一位用户使用，无论用户使用什么浏览器，因为它是始终以渐进增强为原则。

· ****响应式用户界面**** - 适应任何环境：桌面电脑，智能手机，笔记本电脑，或者其他设备。

· ****不依赖网络连接**** - 通过 service workers 可以在离线或者网速极差的环境下工作。

· ****类原生应用**** - 有像原生应用般的交互和导航给用户原生应用般的体验，因为它是建立在 app shell model 上的。

· ****持续更新**** - 受益于 service worker 的更新进程，应用能够始终保持更新。

· ****安全**** - 通过 HTTPS 来提供服务来防止网络窥探，保证内容不被篡改。

· ****可发现**** - 得益于 W3C manifests 元数据和 service worker 的登记，让搜索引擎能够找到 web 应用。

· ****再次访问**** - 通过消息推送等特性让用户再次访问变得容易。

· ****可安装**** - 允许用户保留对他们有用的应用在主屏幕上，不需要通过应用商店。

· ****可连接性**** - 通过 URL 可以轻松分享应用，不用复杂的安装即可运行。

缺点

* 门槛不低。部署的server要求HTTPS，ServiceWorker涉及API众多，须要单独学习
* 浏览器支持不够全面。苹果Safari 短时间内不会支持，5 年计划里可能实施
* 用户体验习惯,网页应用替代原生应用的方式，用户短时间内不适应

## Vue项目中的Unit（单元测试） 和E2E（端到端测试）

前端的测试主要有两种，单元测试和E2E测试，

### Unit测试 (单元测试)：按空间切割，对每个组件进行测试

比如，我要测试日期输入框，那么我编写的测试用例应该包括以下部分：

默认日期是否为当天

当用户选择日期范围，data是否会做相应改变

...

### E2E测试（端对端测试）：按时间切割，对每个流程进行测试

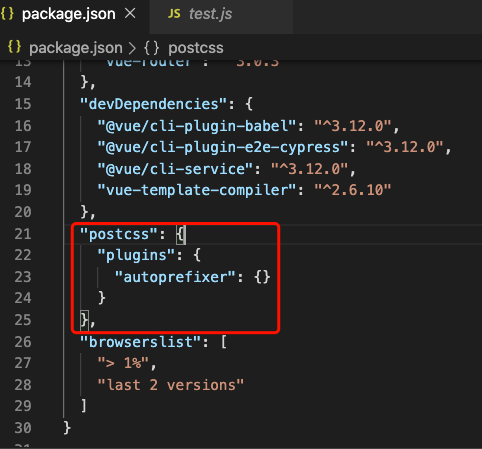
E2E（End To End）即端对端测试，属于黑盒测试，通过编写测试用例，自动化模拟用户操作，确保组件间通信正常，程序流数据传递如预期。

比如，我要测试搜索功能，那么我编写的测试用例应该模拟以下步骤：

1. 打开主页
2. 点击菜单跳转到详情页
3. 输入搜索条件
4. 点击搜索
5. 查看搜索结果是否与预期一致

E2e也称之为自动化测试，即模拟用户进行自动化测试

# **PostCSS**



### Less，sass ，postcss 区别

less sass 是预处理器，用来支持扩充css语法,扩展了原生的东西，它把css作为一个子集,让我们的样式操作能力更加强大，但是却有个问题，那就是----不好保持向后兼容.

这时就出现了postcss,postcss 一种对css编译的工具,postcss 鼓励开发者使用规范的CSS原生语法编写源代码，支持未来的css语法，就像babel支持ES6。

### postcss常见的功能

1 . 使用下一代css语法

2 . 自动补全浏览器前缀

3 . 自动把px代为转换成rem（rem不熟悉的，[点这](https://www.toutiao.com/i6556345600537788932/?group_id=6556345600537788932" \t "/Users/zhanglu/Documents\\x/_blank)）

4 . css 代码压缩等等