**vue生命周期详解**

答： Vue 实例从创建到销毁的过程，就是生命周期。也就是从开始创建、初始化数据、编译模板、挂载Dom→渲染、更新→渲染、卸载等一系列过程，我们称这是 Vue 的生命周期

beforeCreate created beforeMount mounted beforeUpdate updated beforeDestroy destroyed

activated: keep-alive 组件激活时调用

deactivated: keep-alive 组件停用时调用

errorCaptured：当捕获一个来自子孙组件的错误时被调用

1. 实例、组件通过new Vue() 创建出来之后会**初始化事件和生命周期**，然后执行beforeCreate，这时，数据还没有挂载，只是一个空壳，无法访问到数据和真实的dom，一般不做操作
2. 挂载数据，进行初始化事件等，然后执行created，这时可以使用、更改数据，在这更改数据不会触发updated，在这里可以在渲染前倒数第二次更改数据的机会，不会触发其他的钩子函数，一般可以在这里做初始数据的获取
3. 实例或者组件对应的模板，编译模板为虚拟dom放入到render函数中准备渲染，然后执行beforeMount，在这个函数中虚拟dom已经创建完成，马上就要渲染,在这里也可以更改数据，不会触发updated，在这里可以在渲染前最后一次更改数据的机会，不会触发其他的钩子函数，一般可以在这里做初始数据的获取，****该钩子在服务器端渲染期间不被调用****
4. 接下来开始render，渲染出真实dom，然后执行mounted钩子函数，此时，组件已经出现在页面中，数据、真实dom都已经处理好了,事件都已经挂载到el了，可以在这里操作真实dom等事情...
5. 当组件或实例的数据更改之后，会立即执行beforeUpdate，然后vue的虚拟dom机制会重新构建虚拟dom与上一次的虚拟dom树利用diff算法进行对比之后重新渲染，这里适合在更新之前访问现有的 DOM，比如手动移除已添加的事件监听器
6. 当更新完成后，执行updated，数据已经更改完成，dom也重新render完成，可以操作更新后的虚拟dom
7. 当经过某种途径调用$destroy方法后，立即执行beforeDestroy，在这一步，实例仍然完全可用,一般在这里做一些善后工作，例如清除计时器、清除非指令绑定的事件等等
8. destroyed, Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁剩下dom空壳，在这做善后工作也可以

# 第一次页面加载会触发哪几个钩子

答：第一次页面加载时会触发 beforeCreate, created, beforeMount, mounted

**nextTick : 更新数据后立即操作dom**

**父子通信 (直接父子级)**

父->子

<Child-one :parentMessage="parentMessage"></Child-one> 父

props: ['parentMessage'], 子

父组件通过绑定属性把数据传递给子组件，子组件通过props选项接收数据，props可以接受function，也可以做些类型检查

子->父

子组件通过$emit自定义事件并传递数据，父组件使用v-on监听子组件上的自定义事件接收数据

**mounted() {**

**this.$emit('childEvent', { name: 'zhangsan', age: 10 });**

**},**

**Child-one @childEvent="parentMethod"></Child-one>**

**methods: {**

**parentMethod({ name, age }) {**

**console.log(this.parentMessage, name, age);**

**},**

**},**

**props类型**（String,Number,Boolean,Object, Array ,Function）

props可以是数组或对象，用于接收来自父组件的数据

props: ['size', 'myMessage']

props: {  
 // 检测类型  
 height: Number,  
 // 检测类型 + 其他验证  
 age: {  
 type: Number,  
 default: 0,  
 required: true,  
 validator: function (value) {  
 return value >= 0  
 }  
 }

其他：不推荐

因为vue提倡单向数据流，只有父组件交给子组件的数据子组件才有使用的权限

this.$parent this.$refs 不属于数据的传递而是一种主动的查找，利用了父子组件关系传递数据

父->子this.$parent this为子组件实例 获取父组件然后使用父组件中的数据

子->父

可以通过在子组件添加ref属性，然后可以通过ref属性名称获取到子组件的实例

<template>

<div>

<h2>父组件</h2>

<br>

<Child-one ref="child"></Child-one>

</div>

</template>

<script>

import ChildOne from './ChildOne';

export default{

components: {

ChildOne,

},

mounted(){

console.log(this.$refs['child']);

},

};

</script>

<style scoped>

</style>

兄弟组件之间的通信

实例化一个vue实例，相当于一个第三方

let vm = new Vue(); //创建一个新实例

组件他哥

<div @click="ge"></div>

methods: {

ge() {

vm.$emit('blur','sichaoyun'); //触发事件

}

}

组件小弟接受大哥命令

<div></div>

created() {

vm.$on('blur', (arg) => {

this.test= arg; // 接收

});

}

其他不限于**直接父子级**

**Vuex localstorage sessionstorage cookies**

**-----------------------------------------------------------------**

**vue断点调试**

# vue如何实现按需加载配合webpack设置

webpack中提供了require.ensure()来实现按需加载。以前引入路由是通过import 这样的方式引入，改为const定义的方式进行引入。  
不进行页面按需加载引入方式：

import home from '../../common/home.vue'  
进行页面按需加载的引入方式：

const home = r => require.ensure( [], () => r (require('../../common/home.vue')))

# Vue的双向数据绑定原理是什么？或者vue的响应式原理

**vue.js是采用数据劫持，并结合发布者——订阅者的模式：通过Object.defineProperty()来劫持vue中各个属性的setter、getter，在数据变动的时候，再发布消息给订阅者**

**指令：当表达式的值发生改变时，将这个变化反应到DOM上，指令绑定的值都是字符串**

**console.log(app.$data); // 会给data里的属性添加get、set方法**

|  |
| --- |
| vue vue-router vuex vue-loader vue-cli axios rxjs mobx lodash element-ui iview weex 小程序  nuxt webpack gulp css html es6 typescript 数据结构和算法 设计模式 npm yarn  git svn mock echart jquery bootstrap zepto underscore eslint  redux |

# vue组件与自定义元素的关系

|  |
| --- |
| <https://cn.vuejs.org/v2/guide/index.html#与自定义元素的关系>  把data放到响应系统  Vue 将模板编译成虚拟 DOM 渲染函数  值得注意的是只有当实例被创建时 data 中存在的属性才是响应式的。也就是说如果你添加一个新的属性，比如：  vm.b = 'hi'  那么对 b 的改动将不会触发任何视图的更新。  *Object*.freeze()，这会阻止修改现有的属性，也意味着响应系统无法再追踪变化。  *var* obj = {  foo: 'bar'  }  *Object*.freeze(obj)  new Vue({  el: '#app',  data: obj  })  XSS 攻击 |

# 核心思想

|  |
| --- |
| 数据驱动 组件化 虚拟dom  双向数据绑定  view <=*=>* viewModel <=*=>* model  dom vue pojo（原生js对象）  如何实现双向数据绑定  *Object*.defineProperty()  <input type="text", id="a">  <span id="b"></span>  *var* obj = {};  *Object*.defineProperty(obj, 'hello', {  set: *function*(*newVal*) {  document.getElementById('a').value = newVal;  document.getElementById('b').innerHtml = newVal;  }  })  document.addEventListener('keyup', *function*(*e*){  obj.hello = e.terget.value;  }) |

# 创建一个Vue实例

|  |
| --- |
| 通过Vue构造函数创建： <b>*var* vm = new Vue({ // 选项 })</b>  一个Vue应用应由<b>Vue根实例</b>，以及<b>可选的、嵌套的、可复用的</b>组件树构成  所有的 Vue 组件都是 Vue 实例，并且接受相同的选项对象 (*一些根实例特有的选项除外*) |

# 数据与方法

|  |
| --- |
| 一个Vue实例被创建时，会把data对象里的属性加入Vue的响应式系统，属性值改变时，视图会进行重渲染  只有当实例被创建时data中存在的属性才是响应式的，如下  // 添加新的属性,对b的改动不触发视图的更新  // 解决：一开始为空或不存在，后面再赋值  new Vue({  data: {  b: 'hello'  }  })  vm.b = 'hi';  b // hello  *Object*.freeze()会阻止修改现有的属性，也意味着响应系统无法再追踪变化  <p>{{ foo }}</p>  <!-- 这里的 `foo` 不会更新！ -->  <button v-on:click="foo = 'baz'">Change it</button>  *var* obj = {  foo: 'bar'  }  *Object*.freeze(obj)  new Vue({  el: '#app',  data: obj  }) |

# mustache插值

|  |
| --- |
| mustache插值在网络差时，会显示出来，v-text不会  <div v-text="msg">{{msg}}</div>  mustache标签 可以接受表达式的值  {{cents/10}}  {{true? 0: 1}}  {{example.split(",")}}  {{example | filterA | filterB}}  {{example | filter a b}}  {{var a = 10}} 错  {{if(true) return "haha"}} 错 |

# v-text

|  |
| --- |
| 更新部分的 textContent ，需要使用 {{ mustache }} 插值  <span v-text="msg"></span>  等价于  <span>{{msg}}</span> |

# v-html

|  |
| --- |
| 更新元素的 innerHTML,会解析html代码,容易导致 XSS 攻击  <span v-html="html"></span> |

# v-show

|  |
| --- |
| 注意: v-show 不支持<template>语法, 也不支持 v-else  据表达式之真假值，切换元素的 display CSS 属性  <input type="button" value="按钮" @click="flag=!flag">  <span v-show="flag"> HTML </span>  <span v-else> CSS </span>  data:{  flag:true  } |

# v-if

|  |
| --- |
| 1.在切换时,元素及它的数据绑定 / 组件被销毁并重建。如果元素是 <template> ，将提出它的内容作为条件块。  2.v-else 元素必须紧跟在 v-if 元素或者 v-else-if的后面——否则它不能被识别  <span v-if="flag1 = num>11?true:false">HTML</span>  <span v-else-if="flag2 = num>12?true:false">CSS</span>  <span v-else>JS</span>  data:{  num:120,  }  <template v-if="ok">  <h1>HTML</h1>  <p>CSS</p>  <p>JS</p>  </template> |

# v-cloak

|  |
| --- |
|  |

# v-bind

|  |
| --- |
| v-bind:class  值为数组时，数组的值可以是对象(多个)或字符串(多个)，推荐在data写好类样式，再赋值,  值为对象时，也推荐在data写好，再赋值  .red {*color*:red;}  .font {*font-size*:24px;}  .bg {*background*:blue;}  <span v-bind:class="[red,font]">HTML</span> <!--:class值为数组-->  <span v-bind:class="arr">HTML</span> <!--:class值为数组-->  <span v-bind:class="[json,json1]">HTML</span> <!--:class值为数组，数组值是对象-->  <span v-bind:class="arr1">HTML</span> <!--:class值为数组，数组值是对象-->  <span v-bind:class="{red:true, font:false}">HTML</span> <!--:class值为对象-->  <span v-bind:class="{red:aaa, font:bbb}">HTML</span> <!--:class值为对象-->  <span v-bind:class="json">HTML</span> <!--:class值为对象,不能多个对象-->  选中isOk则为true  <input type="checkbox" id="isTrue" v-model="isOk">  <label for='isTrue'>isOk={{isOk}}</label>  <span :class="isOk?classA:classB">绑定class中的三元表达式判断</span>  data:{  red:"red",  font:"font",  arr:["red","font"],  aaa:true,  bbb:false  json:{red:true,font:false},  json1:{bg:true},  arr1:[  {red:true, font:false},  {bg:true}  ]  }  ======================================================================================  v-bind:style  值为数组时，数组的值只能是对象(多个)，推荐在data写好类样式，再赋值,  值为对象时，也推荐在data写好，再赋值  注意：复合样式，采用驼峰命名法  <span v-bind:style="{color:'red',fontSize:'24px'}">HTML</span> <!--值为对象-->  <span v-bind:style="{color:'red',fontSize:fontSize+'px'}">HTML</span> <!--值为对象-->  <span v-bind:style="json">HTML</span> <!--值为对象-->  <span v-bind:style="[json,json1]">HTML</span> <!--值为数组-->  data:{  fontSize:24,  json:{color:'red',fontSize:"24px"},  json1:{background:"blue"}  }  ===================================================================================  注意：  <button v-bind:disabled="flag">Button</button> <!—flag为false的时，该属性会被移除-->  <img src="{{url}}"/> <!—效果能出来，但是会报一个404错误-->  <img :src="url"/>  <img v-bind:src="url" v-bind:width="width" :title="title"/>  data:{  url:'lyf.jpg',  width:"200px",  title: "vue"  } |

# v-for

|  |
| --- |
| 可以用带有 v-for 的 <template> 标签来渲染多个元素块  v-for 默认行为试着不改变整体，而是替换元素。迫使其重新排序的元素,您需要提供一个 key 的特殊属性,  数组或对象有重复的值时要添加该值才输出  vue2.0里面默认就可以添加重复数据,去掉了隐式一些变量($index $key)  track-by="id" 变成 :key="index" 提高性能  之前:v-for="(index,val) in array"  现在:v-for="(val,index) in array"  <li v-for="(val,index) in arr" :key="index">{{val}} {{index}}</li>  <li v-for="(val,key,index) of obj">{{val}} {{key}} {{index}}</li>  <button @click="add">按钮</button>  data: {  arr:['apple','banana','orange','pear','pear'],  obj:{a:'apple',b:'banana',c:'orange'}  },  methods: {  add() {  this.arrs.push('vue2') //vue2可以无限添加  }  } |

|  |
| --- |
| Push() pop() shift() unshife() splice() sort() reverse() |

**v-for 练习**

|  |
| --- |
| <li v-for="item in sortItems">{{item}}</li>  <li v-for="(student,index) in sortStudent">  {{index}}：{{student.name}} - {{student.age}}  </li>  data:{  items:[20,23,18,65,32,19,5,56,41],  students:[  {name:'jspang',age:32},  {name:'Panda',age:30},  {name:'PanPaN',age:21},  {name:'King',age:45}  ]  },  computed:{  sortItems:*function*(){  return this.items.sort(sortNumber);  },  sortStudent:*function*(){  return sortByKey(this.students,'age');  }  }  //数组对象方法排序:  *function* sortByKey(*array*,*key*){  return array.sort(*function*(*a*,*b*){  *var* x=a[key];  *var* y=b[key];  return ((x<y)?-1:((x>y)?1:0));  });  }  *function* sortNumber(*a*,*b*){  return a-b  } |

# v-ref

# v-cloak

|  |
| --- |
| Vue遇到v-cloak时，会先去掉，等实例准备完毕才显示  v-cloak 防止闪烁, 比较大段落v-text v-html  <style>  [v-cloak]{  *display*: none;  }  </style>  <div v-cloak>HTML</div> |

# v-pre

|  |
| --- |
| <span v-pre>{{msg}}</span> 原样输出 |

# v-once

|  |
| --- |
| 只渲染元素和组件一次  <input type="text" v-model="msg">  <div>{{msg}}</div>  <div v-once>{{msg}}</div> 继续输入不会再渲染  <my-component v-once :comment="msg"></my-component> |

# v-model

|  |
| --- |
| 修饰符：lazy number debounce  <input type="text" v-model="message">  <input type="text" v-model.lazy="message"> 失去焦点时才更新  <input type="text" v-model.number="message"> 开始输入是string，后面再输入字符串，number无效  <input type="text" v-model.trim="message"> |

# v-slot

技术胖

|  |
| --- |
| v-slot旧版本  slot:位置、槽口，作用: 占个位置  <div id="app">  <jspang>  <span slot="bolgUrl">{{jspangData.bolgUrl}}</span>  <span slot="netName">{{jspangData.netName}}</span>  <span slot="skill">{{jspangData.skill}}</span>  </jspang>  </div>  <template id="tmp">  <div>  <p>博客地址：<slot name="bolgUrl">默认显示我</slot></p>  <p>网名：<slot name="netName">默认显示我</slot></p>  <p>技术类型：<slot name="skill">默认显示我</slot></p>  </div>  </template>  *var* jspang={  template:'#tmp'  }  *var* app=new Vue({  el:'#app',  data:{  jspangData:{  bolgUrl:'http://jspang.com',  netName:'技术胖',  skill:'Web前端'  }  },  components:{  "jspang":jspang  }  }) |

# v-on

## 鼠标事件

|  |
| --- |
| 缩写 @click  click/mouseout/mouseover/dblclick/mousedown.....  <input type="button" v-on:click="add" value="按钮"/>  <input type="button" v-on:click="add($event,10)" value="按钮"/> 传参  add:*function*(*ev*,*num*){  *console*.log(ev,num)  } |

## 键盘事件

|  |
| --- |
| 常用键:  keyup, keydown 区别，keydown按下就触发  @keyup.13 或 @keyup.enter 回车  @keyup/keydown.left  @keyup/keydown.right  @keyup/keydown.up  @keyup/keydown.down  <input type="text" @keyup="show($event)"/>  <input type="text" @keyup.enter="show($event)"/>等价于  <input type="text" @keyup.13="show($event)"/>  <input type="text" @keydown.c="show($event)"/> 等价于  <input type="text" @keydown.67="show($event)"/>  show:*function*(*ev*){  *console*.log(ev.keyCode);  } |

## 冒泡 阻止默认行为

|  |
| --- |
| <input type="text" @click.stop="show($event)"/><!-- vue阻止冒泡 -->  <input type="text" @contextmenu.prevent="show($event)"/> <!-- 阻止默认行为(默认事件) -->  <form v-on:submit.prevent="onSubmit"></form> <!-- 提交事件不再重载页面 -->  show:*function*(*ev*){  ev.cancelBubble = true;//js阻止冒泡  ev.preventDefault(); // js阻止默认行为  *console*.log(ev.keyCode);  } |

## 自定义键盘信息

|  |
| --- |
| <input type="text" @keyup.ctrl="change($event)">  Vue.directive('on').keyCodes.ctrl=17; //等价于  Vue.config.keyCodes.ctrl=13;  change(*ev*){  *console*.log(ev);  }  =========================================== |

|  |
| --- |
| Vue1.0  <!-- 要自定义指令才会输出 -->  <input type="text" @keydown.ctrl="show1"/>  <input type="text" @keydown.myenter="show2 | debounce 2000"/>  Vue.directive('on').keyCodes.ctrl=17;  Vue.directive('on').keyCodes.myenter=13;  methods:{  show1:*function*(*ev*){  *console*.log(ev.keyCode);  },  show2:*function*(*ev*){  *console*.log(ev.keyCode);  }  } |

# 全局配置

## Vue.config

|  |
| --- |
| Vue.config 是一个对象，包含 Vue 的全局配置。可以在启动应用之前修改下列属性  IMG_256 |

## silent

|  |
| --- |
| <div id="app">  <span>{{msg}}</span>  </div>  // Vue.config.silent = true; // 取消 Vue 所有的日志与警告。  *var* vm = new Vue({  el: '#app',  data: {  //msg:'hello world' // 使用了未定义的data属性msg  }  });  IMG_256 |

## optionMergeStrategies

|  |
| --- |
| // 全局注册混入对象,将会影响到 所有 之后创建的 Vue 实例  *let* globalMinin = Vue.mixin({  created: *function* () {  *var* haha = this.$options.haha || 'xixi'  *console*.log(haha)  },  myOption: {  getters: 'getToVal',  state: 'staToVal',  actions: 'actToVal'  }  })  *const* merge = Vue.config.optionMergeStrategies.computed // 使用computed合并策略  // 将globalMinin的myOption和myVue的myOption按照 自定义选项合并策略myOption进行合并  Vue.config.optionMergeStrategies.myOption = *function* (*toVal*, *fromVal*, *vm*) {  *console*.dir(toVal)  *console*.dir(fromVal)  *console*.dir(vm)  *console*.dir('-----------------')  if (!toVal) return fromVal  if (!fromVal) return toVal  return {  getters: merge(toVal.getters, fromVal.getters),  state: merge(toVal.state, fromVal.state),  actions: merge(toVal.actions, fromVal.actions)  }  }  /\* eslint-disable no-new \*/  *let* myVue = new Vue({  el: '#app',  haha: 'haha',  myOption: {  getters: 'getFromVal',  state: 'staFromVal',  actions: 'actFromVal'  },  router,  store,  mock,  components: { App },  template: '<App/>'  })  IMG_256 |

## devtools

|  |
| --- |
| // 务必在加载 Vue 之后，立即同步设置以下内容  Vue.config.devtools = true  /\* 配置是否允许 vue-devtools 检查代码。开发版本默认为 true，生产版本默认为 false。  生产版本设为 true 可以启用检查。\*/  IMG_256 |

## errorHandler

|  |
| --- |
| // 指定组件的 渲染和观察期间 未捕获错误的处理函数。这个处理函数被调用时，可获取错误信息和 Vue 实例。  Vue.config.errorHandler = *function* (*err*, *vm*, *info*) {  // handle error  // `info` 是 Vue 特定的错误信息，比如错误所在的生命周期钩子  // 只在 2.2.0+ 可用  *console*.log('---------------err-----------------');  *console*.log(err);  *console*.log('---------------info---------------');  *console*.log(info);  *console*.log('---------------vm------------------');  *console*.log(vm);  *console*.log('---------------arguments------------------');  *console*.log(arguments)  }  *var* vm = new Vue({  el: '#app',  data: {  msg:'hello world'  },  created: *function* () {  *var* x = '';  if(x === '') throw "值为空";  else if(isNaN(x)) throw "不是数字";  else if(x>10) throw "太大";  /\*try {  if(x === '') throw "值为空";  else if(isNaN(x)) throw "不是数字";  else if(x>10) throw "太大";  } catch(err) {  console.log('err: ' + err);  }\*/  }  });  IMG_256 |

## warnHandler

|  |
| --- |
| <div id="app">  <span>{{msg}}</span>  </div>  // 为 Vue 的运行时警告赋予一个自定义处理函数。注意这只会在开发者环境下生效，在生产环境下它会被忽略。  // msg trace 跟[Vue warn] 一样  Vue.config.warnHandler = *function* (*msg*, *vm*, *trace*) {  *console*.log('---------------msg-----------------');  *console*.log(msg);  *console*.log('---------------trace---------------');  *console*.log(trace); // `trace` 是组件的继承关系追踪  *console*.log('---------------vm------------------');  *console*.log(vm);  }  *var* vm = new Vue({  el: '#app',  data: {  //msg:'hello world' // 使用了未定义的data属性msg  }  });  IMG_256 |

## ignoredElements

## keyCodes

|  |
| --- |
| <input type="text" @keydown.media-play-pause="change($event)">  // 按下keyCose为179键才触发change  Vue.config.keyCodes = {  v: 86,  f1: 112,  // camelCase 不可用  mediaPlayPause: 179,  // 取而代之的是 kebab-case 且用双引号括起来  "media-play-pause": 179,  up: [38, 87]  }  *var* vm = new Vue({  el: '#app',  data: {  msg:'hello world'  },  methods:{  change:*function*(*ev*){  *console*.log(ev.keyCode);  *console*.log('改变了');  }  }  }); |

## performance

|  |
| --- |
| // <my-aaa></my-aaa>  // 设置为 true 以在浏览器开发工具的 性能/时间线 面板中启用对组件初始化、编译、渲染和打补丁的性能追踪。  // 只适用于开发模式和支持 performance.mark API 的浏览器上。  Vue.config.performance = true;  //全局组件  Vue.component('my-aaa',{  template:'<h3 @click="change">{{msg}}</h3>',  data(){  return { msg:'html'}  },  methods:{  change() {this.msg='javascript'; }  }  })  *var* vm = new Vue({  el: '#app',  data: {  msg:'hello world'  }  });  查看自定义组件<my-aaa></my-aaa>的各个阶段花费的时间：  IMG_256 |

## productionTip

|  |
| --- |
| Vue.config.productionTip = false; // 设置为 false 以阻止 vue 在启动时生成生产提示。  IMG_256  IMG_256 |

# 全局 API

## Vue.extend

|  |
| --- |
| <author></author>  <div id="author"></div>  /\*  扩展实例构造器，创建一个“子类”，data是函数，返回一个对象,经常和组件使用  \*/  *var* authorExtend = Vue.extend({  template:"<p><a :href='authorUrl'>{{authorName}}</a></p>",  data:*function*(){  return{  authorName:'xiaoming',  authorUrl:'http://www.com'  }  }  });  *console*.log(new authorExtend());  new authorExtend().$mount('author');  new authorExtend().$mount('#author'); |

案例

|  |
| --- |
| <shi-peng></shi-peng>  <div id="shi-peng"></div>  /\*  Vue.extend 返回的是一个“扩展实例构造器”,也就是预设了 "部分" 选项的Vue实例构造器。  经常服务于Vue.component用来生成组件，理解为在模板中遇到该组件名称作为标签的自定义元素时，  会自动调用“扩展实例构造器”来生产组件实例，并挂载到自定义元素上,  但是他没有传递任何参数，只是个静态标签。  \*/  *var* Profile = Vue.extend({  template:'<p @click="change">{{msg}}</p>',  data:*function*(){  return {  msg:'html'  }  },  methods:{  change: *function*() {  this.msg = 'javascript'  }  }  })  // 直接挂载时click不可以  new Profile().$mount('shi-peng');  new Profile().$mount('#shi-peng'); //挂载到id为shi-peng的div上  //全局组件 此时click可用  //Vue.component('shi-peng',Profile);  IMG_256 |

技术胖

|  |
| --- |
| {{message}}  <button @click="add">add</button>  // extends 只能拓展构造器里原生的方法  *var* bbb={  created:*function*(){  *console*.log("我是被扩展出来的"); // 执行  },  methods:{  add:*function*(){  *console*.log('我是被扩展出来的方法！'); // 不执行  }  }  };  *var* app=new Vue({  el:'#app',  data:{  message:'hello Vue!'  },  methods:{  add:*function*(){  *console*.log('我是原生方法'); // 点击执行  }  },  extends:bbb, // 只能是对象或方法，不能是数组  delimiters:['${','}'] // ${num}  }) |

## Vue.nextTick

|  |
| --- |
| <https://www.cnblogs.com/first-time/p/7087830.html>  <https://www.cnblogs.com/xujiazheng/p/6852124.html>  <https://www.cnblogs.com/douyaer/p/7650140.html>  <shi-peng></shi-peng>  *var* Profile = Vue.extend({  template:'<p >{{msg}}</p>',  data:*function*(){  return {  msg:'htmlxxxx'  }  },  created(){  //this.change();  },  beforeMount() {  //this.change();  },  mounted () {  //this.change();  },  beforeUpdate() {  this.change();  },  methods:{  change: *function*() {  this.msg = 'javascriptxxx';  *console*.log(this.$el.textContent); // => '没有更新'  //console.log(this.msg); // => '没有更新'  this.$nextTick(*function* () {  *console*.log(this.$el.textContent); // => '更新完成'  //console.log(this.msg); // => '更新完成'  })  }  }  }) |

## Vue.set

技术胖

|  |
| --- |
| <div id="app">  <p>{{count}}</p>  <ul>  <li v-for="value in arr">{{value}}</li>  </ul>  </div>  <button onclick="add()">外部添加</button>  *function* add() {  /\*  Vue.set(outData,'count',4);  app.count++;  outData.count++;  **使用外部outData时，以上3种方法都可以让基本类型修改成功**  \*/  **// 有以上其中一句都可以使**app.arr[1]='ddd' **成功，因为触发了虚拟dom更新**  **// 只有该句时，使用下标修改数组的值时，不成功，js的限制**  //app.arr[1]='ddd';  Vue.set(app.arr,1,'ddd'); // 成功  }  // data在外部声明  *var* outData={  count:1,  arr:['aaa','bbb','ccc']  };  *var* app=new Vue({  el:'#app',  data:outData //引用外部数据  }) |

案例

|  |
| --- |
| <http://jspang.com/2017/03/14/vue2_02/#3vueset>  <https://cn.vuejs.org/v2/guide/reactivity.html>  Vue.set(target,key,value)普通的情况下对Vue实例里面的数据进行更改，数据改掉了，  但是呈现在页面的视图并没有发生变化，所以借用该方法视图也会跟着刷新    <li v-for="item in arr">{{item}}</li>  var app=new Vue({  el:"#app",  data:{  arr:['北京','上海','天津','重庆']  }  });  app.arr[0]='北京天安门';// 不会刷新  Vue.set(app.arr,0,'北京天安门'); // set方式视图会刷新 |

## Vue.delete

|  |
| --- |
| <https://blog.csdn.net/sinat_17775997/article/details/75950335>  <https://segmentfault.com/a/1190000011512960>  Vue.delete(target,key)用法和原理与set添加元素是一样的道理  *var* data = {  msg:'xcssdsds',  arr:['aaa','bbb','ccc']  }  Vue.delete(data, 'msg');  Vue.delete(data.arr, 0);  *var* vm = new Vue({  el: '#app',  data: data,  }); |

## Vue.directive自定义指令-属性指令

|  |
| --- |
| 作用：验证数据  <span v-color>HTML</span>  <span v-color="'red'">HTML</span> <!-- 传入的是字符串，双引号包裹单引号 -->  // 自定义指令color要添加前缀v-color,my-color => v-my-color  Vue.directive('color',*function*(*el*,*binding*){  binding.value = binding.value ? binding.value : 'blue';  el.style.background = binding.value;  });  // 第二参数是对象  Vue.directive('color',{  bind:*function*(*el*,*binding*,*vnode*,*oldVnode*) {  binding.value = binding.value ? binding.value : 'blue';  el.style.background = binding.value;  *console*.log('bind',el,binding,vnode,oldVnode);  },  inserted:*function*(*el*,*binding*) {  *console*.log('inserted',el,binding);  },  update:*function*() {  *console*.log('update');  },  componentUpdated:*function*() {  *console*.log('componentUpdated');  },  unbind:*function*() {  *console*.log('unbind');  },  }); |

技术胖directive

|  |
| --- |
| <div id="app">  <div v-mydirective="color" id="aaa">{{num}}</div>  <button @click='jia'>加分</button>  <button onclick='unbind()'>解绑</button>  </div>  *function* unbind(){  app.$destroy();  }  //自定义指令  Vue.directive('mydirective',{  bind:*function*(*el*,*binding*,*vnode*){//被绑定  /\*\*  var s=JSON.stringify;  el.innerHTML =  'name:' + s(binding.name) +'<br>' +  'value:' + s(binding.value) +'<br>' +  'expression:' + s(binding.expression) +'<br>' ;  \*\*/  el.style='color:'+binding.value;  *console*.log('1 - bind');  },  inserted:*function*(){//绑定到节点  *console*.log('2 - inserted');  },  update:*function*(){//组件更新  *console*.log('3 - update'); // 元素的值更新或dom更新是调用  },  componentUpdated:*function*(){//组件更新完成  *console*.log('4 - componentUpdated');  },  unbind:*function*(){//解绑  *console*.log('5 - unbind');  }  })  *var* app=new Vue({  el:'#app',  data:{  color:'green',  num:10  },  methods:{  jia:*function*(){  this.num++;  }  }  }) |

## Vue.directive自定义指令-元素指令

|  |
| --- |
| <style>  my-red {  *width*: 100px;  *height*: 100px;  *background*: blue;  *display*:block;  }  </style>  <my-red></my-red>  // vue1.0注册或取得一个全局自定义元素指令  Vue.elementDirective('my-red',{ //不支持function语法  bind:*function*(){  this.el.style.background='black'; //注释该句为blue  }  }) |

## Vue.directive自定义指令-拖拽

|  |
| --- |
| <div v-drag :style="{width:'100px',height:'100px',background:'blue',  position:'absolute',right:0,top:0}"></div>  <div v-drag :style="{width:'100px', height:'100px', background:'red',  position:'absolute', left:0, top:0}"></div>  Vue.directive('drag',*function*(*el*){  el.onmousedown = *function*(*ev*){  *var* disX = ev.clientX-el.offsetLeft;  *var* disY = ev.clientY-el.offsetTop;  document.onmousemove=*function*(*ev*){  *var* l=ev.clientX-disX;  *var* t=ev.clientY-disY;  el.style.left=l+'px';  el.style.top=t+'px';  };  document.onmouseup=*function*(){  document.onmousemove=null;  document.onmouseup=null;  };  };  }); |

## Vue.filter过滤器vue1.0

### 常用过滤器

|  |
| --- |
| 常用过滤器  {{'welcome'|uppercase}} 转大写  {{'WELCOME'|lowercase}} 转小写  {{'WELCOME'|lowercase|capitalize}} 转小写,再首字母大写  {{12|currency}} $12  {{12|currency '￥'}} ￥12  {{json | json}} 等价于JSON.stringify |

### debounce

|  |
| --- |
| debounce 配合事件，延迟执行，lodash \_.debounce(fn,时间)  <input type="text" @keyup="show | debounce 2000">  methods:{  show:function(){  console.log(123);  }  } |

### limitBy

|  |
| --- |
| limitBy数据配合使用过滤器: limitBy  限制几个 limitBy 取几个 从哪开始  filterBy 过滤数据  orderBy 排序 1/-1 1 -> 正序 -1 -> 倒序 按照名字排  <li v-for="val in arr | limitBy 2">{{val}}</li>  <li v-for="val in arr | limitBy 2 arr.length-2">{{val}}</li>  data:{  arr:[1,2,3,4,5]  } |

### filterBy

|  |
| --- |
| filterBy  <input type="text" v-model="char">  <li v-for="val in arr | filterBy char">{{val}}</li>  data:{  arr:['width','height','background','orange'],  char:''  } |

### orderBy

|  |
| --- |
| orderBy  <input type="text" v-model="char"/>  <li v-for="val in arr | orderBy char">{{val}}</li> 排序  <li v-for="val in arr | orderBy -1">{{val}}</li>  <li v-for="val in arr | orderBy 'o'">{{val}}</li>  data:{  arr:['width','height','background','orange'],  char:''  } |

## Vue.filter过滤器vue2.0

|  |
| --- |
| 过滤器用在两个地方：**mustache 插值和 v-bind表达式**, 不能用在v-model、v-html、v-text  为了在其他指令中实现更复杂的数据变换，应该使用计算属性 |

### 自定义过滤器

|  |
| --- |
| ***model ->过滤 -> view***  vue2.0，删除全部内置过滤器,推荐使用工具库lodash \_.debounce(fn,200)  {{msg}} // 2.0可以直接解析json  data:{msg:{a:1,b:2,c:3} }  自定义过滤器——还保留，但是,自定义过滤器传参改变  之前: {{msg | toDou '12' '5'}}  现在: {{msg | toDou('12','5')}}  {{num | toDou(1,2)}}  Vue.filter('toDou',*function*(*num*,*a*,*b*){  return num<10? '0'+num:''+num; //转化成字符串  })  new Vue({  data:{  num:90  }  }).$mount('#app'); // vm.$mount('#box'); //手动挂载 |

### 时间转化器

|  |
| --- |
| {{num | date}}  Vue.filter('date',*function*(*input*){  *var* oDate = new Date(input);  return oDate.getFullYear()+'-'+(oDate.getMonth()+1)+'-'+oDate.getDate()+  ' '+oDate.getHours()+':'+oDate.getMinutes()+':'+oDate.getSeconds();  });  data:{  num:*Date*.now() //1970 年 1 月 1日午夜与当前日期和时间之间的毫秒数  }, |

### 过滤html标记

|  |
| --- |
| {{msg | fiterHtml}}  Vue.filter('fiterHtml',*function*(*input*){  return input.replace(/<[^<]+>/g,'');  });  data:{  msg:'<strong>welcome</strong>'  } |

### 双向过滤器vue1.0

|  |
| --- |
| {{{msg | filterHtml}}}  Vue.filter('filterHtml',{  read:*function*(*input*){ //model-view  return input.replace(/<[^<]+>/g,'');  },  write:*function*(*val*){ //view -> model  *console*.log(val);  return val;  }  });  data:{  msg:'<strong>welcome</strong>'  }, |

## Vue.component

### 全局组件

|  |
| --- |
| <aaa></aaa>  var Aaa = Vue.extend({  data(){ **//必须是函数形式,函数必须返回一个对象(json)**  return { msg:'刘亦菲' }  },  methods:{  **change**(){ this.**msg** = '大魔王'}  },  template:'<h3 @click="change">{{msg}}</h3>'  // template:'<h3>刘亦菲</h3>' });  Vue.**component**('aaa',Aaa); //全局组件  var a = new Aaa(); **console**.log(Aaa); // function VueComponent(options){...} **console**.log(a); // VueComponent{...}  =========================================================================================  <my-aaa></my-aaa>  Vue.**component**('my-aaa',{ //全局组件  **template**:'<h3 @click="change">{{msg}}</h3>',  data(){  return { msg:'刘亦菲'}  },  methods:{  **change**() {this.**msg**='大魔王'; }  } }) |

### 局部组件

|  |
| --- |
| <aaa></aaa>  <my-aaa></my-aaa>  var Aaa = Vue.extend({  **template**:'<h3>{{msg}}</h3>',  data(){  **// data必须是函数的形式，函数必须返回一个对象(json)**  return {  msg:'刘亦菲'  }  } });  var vm=new Vue({  **el**:'#app',  **components**:{ //局部组件  **aaa**:Aaa,  **'my-aaa'**: Aaa ***// 有 – 连接时，是字符串***  }  });  =============================================================================  <my-aaa></my-aaa>  var vm=new Vue({  **el**:'#box',  components:{ //局部组件  'my-aaa':{  data(){  return { msg:'welcome vue'}  },  methods:{  **change**(){ this.**msg**='changed'; }  },  template:'<h2 @click="change">标题2->{{msg}}</h2>'  }  } }); |

|  |
| --- |
| <world></world>  *var* city={  template:`<div>this is city!</div>`  }  *var* world={  template:`<div>  <p>局部注册的- &lt;world&gt;</p>  <city></city>  </div>`,  components:{  "city":city  }  }  *var* app=new Vue({  el:'#app',  data:{  message:'hello Vue!'  },  components:{  "world":world  }  }) |

### 组件模板

|  |
| --- |
| 模板定义有4种：  script标签模板  template 标签模板  在构造器的template属性里定义：template:` <h1>我是选项模板</h1>`  .vue文件里定义 |

|  |
| --- |
| <my-aaa></my-aaa>  <!--script标签模板-->  <script type="x-template" id="aaa" src="demo.js"></script> 引入外部文件，形成模板  <script type="x-template" id="aaa">   <h2 @click="change">{{msg}}</h2>  <ul>  <li v-for="val in arr">{{val}}</li>  </ul> </script>  ***或***  <!--template 标签模板 -->  <**template** id="aaa">  <h2 @click="change">{{msg}}</h2>  <ul>  <li v-for="val in arr">{{val}}</li>  </ul> </**template**>  new Vue({  **el**:"#app",  components:{  'my-aaa':{  data(){  return {  msg:'刘亦菲',  arr:['apple','banana','orange']  }  },  methods:{  **change**(){  this.**msg** = '大魔王';  }  },  template:"#aaa"  }  } }) |

### vue2.0-组件模板

|  |
| --- |
| <div id="app">  <my-aaa></my-aaa>  </div>  <!--  vue2.0必须包含一个根标签,vue1.0可以不包含根标签，在每个组件模板，不在支持片段代码  否则报错:Component template should contain exactly one root element  -->  <template id="aaa">  <div>  <h3>刘亦菲</h3>  <strong>大魔王</strong>  </div> </template>  Vue.**component**('my-aaa',{  //template:'<h3>刘亦菲</h3><strong>大魔王</strong>'  **template**:'#aaa' }); |

### vue2.0-组件定义

|  |
| --- |
| Vue.extend 这种方式，在2.0里面有，但是有一些改动，这种写法，即使能用，咱也不用——废弃  <div id="app">  <my-aaa></my-aaa> </div> <template id="aaa">  <div>  <h3>刘亦菲</h3>  <strong>大魔王</strong>  </div> </template>  var Home= {  **template**:"#aaa",  }; Vue.component('my-aaa',Home); // 全局组件  new Vue({  **el**:'#app',  **components**:{  **'my-aaa'**:Home //局部组件  }  })  =========================================================  <hello></hello>  <helloworld></helloworld>  Vue.component('hello',{  template:`<div>全局化注册的：&lt;hello&gt;标签</div>`  });  *var* helloworld = { template:`<div>局部注册的：&lt;helloworld&gt;标签</div>`}  *var* app=new Vue({  el:'#app',  data:{  message:'hello Vue!'  },  components:{  "helloworld":helloworld  }  }) |

### 动态组件

|  |
| --- |
| <input type="button" @click="a='aaa'" value="aaa组件"/> <input type="button" @click="a='bbb'" value="bbb组件"/> <component :is="a"></component>  new Vue({  **el**:"#app",  **data**:{  **a**:'aaa'  },  **components**:{  **'aaa'**:{  **template**:'<h2>我是aaa组件</h2>'  },  **'bbb'**:{  **template**:'<h2>我是bbb组件</h2>'  }  } }) |

|  |
| --- |
| <component v-bind:is="who"></component>  <button @click="changeComponent">changeComponent</button>  *var* componentA={  template:`<div style="color:red;">I'm componentA</div>`  }  *var* componentB={  template:`<div style="color:green;">I'm componentB</div>`  }  *var* componentC={  template:`<div style="color:pink;">I'm componentC</div>`  }  *var* app=new Vue({  el:'#app',  data:{  who:'componentA'  },  components:{  "componentA":componentA,  "componentB":componentB,  "componentC":componentC,  },  methods:{  changeComponent:*function*(){  if(this.who=='componentA'){  this.who='componentB';  }else if(this.who=='componentB'){  this.who='componentC';  }else{  this.who='componentA';  }  }  }  }) |

## Vue.use

## Vue.mixin

|  |
| --- |
| {{ message }}  // 混入2个用途，  // 1.构造器写好了，项目写好了，临时需求，不想污染现有的构造器  // 2.公用的方法  //全局混入  Vue.mixin({  created:*function*(){  *console*.log('我是全局被混入的');  }  })  //全局部混入  *var* aaa={  created:*function*(){  *console*.log('我是外部被混入的');  }  }  *var* app=new Vue({  el:'#app',  data:{  message:'hello Vue!'  },  created:*function*(){  *console*.log('我是原生的');  },  mixins:[aaa] // 只能是数组  }) |

案例

|  |
| --- |
| <http://jspang.com/2017/03/26/vue3/5/>  // 全局混入  Vue.mixin({  created(){  this.change1();  },  beforeMount() {  this.change2();  },  mounted () {  this.change3();  },  beforeUpdate() {  this.change4();  },  data: *function* () {  return {  message: 'hello',  foo: 'abc',  arr:['a','b','c']  }  },  methods:{  hello: *function*() {  *console*.log('这是混合器添加的hello方法');  },  lsit:*function*(){  *console*.log('这是混合器添加的lsit方法')  },  say: *function*() {  *console*.log('这是混合器添加的say方法')  },  change1: *function*() {  *console*.log('change1');  },  change2: *function*() {  *console*.log('change2');  },  change3: *function*() {  *console*.log('change3');  },  change4: *function*() {  *console*.log('change4');  }  }  })  // 注册组件 返回的Profile 是个构造函数  *var* Profile = Vue.extend({  template:'<p >{{msg}}</p>',  data:*function*(){  return {  message: 'xhello',  foo: 'xabc',  arr:['xa','xb','xc']  }  },  created(){  this.change1();  },  beforeMount() {  this.change2();  },  mounted () {  this.change3();  },  beforeUpdate() {  this.change4();  },  methods:{  change1: *function*() {  *console*.log('xchange1');  },  change2: *function*() {  *console*.log('xchange2');  },  change3: *function*() {  *console*.log('xchange3');  },  change4: *function*() {  *console*.log('xchange4');  }  }  })  Vue.component('shi-peng',Profile);  // 在Profile 的原型对象上定义myapp  *Profile* .prototype.myapp = *function*() {  *console*.log('我是外来入侵者');  }  *console*.log(Profile);  *console*.log(new Profile());  *console*.log(Profile.\_\_proto\_\_);  /\*  全局混入的执行顺序要前于混入和构造器里的方法。  Profile 实例对象上有就调用实例对象上的，没有才去mixin里找  \*/  IMG_256  IMG_256 |

## Vue.compile

## Vue.observable

## Vue.version

# 选项 / 数据

## data

## props

## propsData

|  |
| --- |
| <h1>Vue.extend的propsData 用得很少，仅用于开发环境</h1>  <div id="head2"></div>  *var* CommonHeader = Vue.extend({  template:"<header>我是网站头部 {{ message }} -by {{ username }}</header>",  data:*function* () {  return{  message:"I am message!"  }  },  props:['username']  });  // 向Vue.extend传参propsData属性,实际开发中我们使用的并不多  *var* header = new CommonHeader({  propsData:{  username:"wss"  }  }).$mount("#head2"); |

## computed计算属性

|  |
| --- |
| 需求：num1的值变化，num2的值跟着变化，用计算属性  {{num1}} {{num2}}  data:{  num1:10  },  computed:{  num2:*function*(){ // 是属性，只是写法是函数，不用事件触发就执行,默认调用get  return this.num1+20; //num2的值取决于返回值 业务逻辑代码  },  // 或  num2:{ //一开始只执行get方法，get给num2赋值，以后num2值改变时，先set再get  set:*function*(*val*){  this.num1 = val;  },  get:*function*(){  return this.num1 + 20;  }  }  }  document.onclick = *function*(){  vm.num1 = 100;  } |

技术胖

|  |
| --- |
| <p>{{newPrice}}</p>  <li v-for="item in reverseNews">{{item.title}}-{{item.date}}</li>  *var* newsList=[  {title:'香港或就“装甲车被扣”事件追责 起诉涉事运输公司',date:'2017/3/10'},  {title:'日本第二大准航母服役 外媒：针对中国潜艇',date:'2017/3/12'},  {title:'中国北方将有明显雨雪降温天气 南方阴雨持续',date:'2017/3/13'},  {title:'起底“最短命副市长”：不到40天落马，全家被查',date:'2017/3/23'},  ];  *var* app=new Vue({  el:'#app',  data:{  price:100,  newsList:newsList  },  computed:{  newPrice:*function*(){  return this.price='￥' + this.price + '元';  },  reverseNews:*function*(){  return this.newsList.reverse();  }  }  }) |

## methods

|  |
| --- |
| <div id="app">  {{a}}  <p><button @click="add(2)">add</button></p>  <!-- 构造器里的组件上的事件需添加native -->  <btn @click.native="add(2)"></btn>  </div>  <button onclick="app.add(3)">外部访问构造器里的方法</button>  *var* btn={  template:'<button>外部组件</button>'  }  *var* app=new Vue({  el:'#app',  data:{  a:1  },  methods:{  add:*function*(*num*){  return this.a+=num;  }  },  components:{  "btn":btn  }  }) |

## watch

|  |
| --- |
| <p>今日温度：{{temperature}}°C</p>  <p>穿衣建议:{{this.suggestion}}</p>  <p>  <button @click="add">添加温度</button>  <button @click="reduce">减少温度</button>  </p>  *var* suggestion=['T恤短袖','夹克长裙','棉衣羽绒服'];  *var* app=new Vue({  el:'#app',  data:{  temperature:14,  suggestion:'夹克长裙'  },  methods:{  add:*function*(){  this.temperature+=5;  },  reduce:*function*(){  this.temperature-=5;  }  },  watch:{  temperature:*function*(*newVal*,*oldVal*){  if(newVal>=26) {  this.suggestion=suggestion[0];  }else if(newVal<26 && newVal >=0) {  this.suggestion=suggestion[1];  }else{  this.suggestion=suggestion[2];  }  }  }  })  // 全局的  app.$watch('temperature',*function*(*newVal*,*oldVal*){  if(newVal>=26){  this.suggestion=suggestion[0];  }else if(newVal<26 && newVal >=0) {  this.suggestion=suggestion[1];  }else{  this.suggestion=suggestion[2];  }  }) |

# 选项/其它

## name

## delimiters

|  |
| --- |
| <hello></hello>  <world></world>  Vue.component('hello',{  template:`<div>{this.$parent.message}我是全局化注册的-&lt;hello&gt;</div>`,  //替换插值符号  delimiters:['{','}']  })  *var* world={  template:'<div>{{this.$parent.message}}局部注册的-&lt;world&gt</div>'  }  *var* app=new Vue({  el:'#app',  data:{  message:'hello Vue!'  },  components:{  "world":world  }  }) |

## functional

## model

## inheritAttrs

## comments

# 生命周期10个

|  |
| --- |
| 每个Vue实例在被创建时都要经过一系列的初始化过程(设置数据监听、编译模板、将实例挂载到DOM并在数据变化时更新DOM等。  初始化过程中会运行生命周期钩子的函数，可以在不同阶段添加的代码  生命周期钩子的this上下文指向调用它的Vue实例，固不要在选项属性或回调上使用箭头函数，如下  // 因为箭头函数是和父级上下文绑定在一起的，this不会是如你所预期的Vue 实例  // 经常导致 Uncaught TypeError: Cannot read property of undefined 或  // Uncaught TypeError: this.myMethod is not a function 之类的错误  created: () *=>* *console*.log(this.a) 或  vm.$watch('a', *newValue* *=>* this.myMethod()) |

|  |
| --- |
| <div id="app">  <input type="button" value="更新数据" @click="update">  <input type="button" value="销毁组件" @click="destroy">  </div>  <button onclick="app.$destroy()">销毁</button>  {{msg}}  new Vue({  el:'#app',  data:{  msg:'welcome vue2.0'  },  methods:{  update(){  this.msg='大家好';  },  destroy(){  this.$destroy();  }  },  beforeCreate(){  *console*.log('组件实例刚刚被创建,属性都没有');  },  created(){  *console*.log('实例已经创建完成，属性已经绑定');  },  beforeMount(){  *console*.log('模板编译之前');  },  mounted(){  *console*.log('模板编译完成，代替之前ready ');  },  beforeUpdate(){  *console*.log('组件更新之前，数据更新前');  },  updated(){  *console*.log('组件更新完毕，被更新后');  },  activated:*function*(){  *console*.log('activated');  },  deactivated:*function*(){  *console*.log('deactivated');  },  beforeDestroy(){  *console*.log('组件销毁之前');  },  destroyed(){  *console*.log('组件销毁之后');  }  }); |

# vue实例属性

|  |
| --- |
| vue实例属性可以在new Vue() 构造函数外访问其属性和方法  <https://cn.vuejs.org/v2/api/#实例属性>  注意：vm.$options 获取自定义属性  *var* vm = new Vue({  el:"#app",  num :12, // 自定义属性  show:*function*(){ // 自定义方法  *console*.log(123);  }  })  vm.$el.style.background='red'; // vm.$el等价于 el  vm.$options.show(); //123  *console*.log(vm.$options.num); // 12 |

# vue实例方法/数据

## vm.$watch监听数据变化

|  |
| --- |
| 根据  <input type="text" v-model="num">  <input type="text" v-model="json.name">  <div>{{num}} </div>  <div>{{json}}</div>  data:{  num:10,  json:{name:'HTML',age:18}  },  vm.$watch('num',*function*(*newVal*,*oldVal*){  *console*.log(newVal,oldVal);  })  vm.$watch('json',*function*(*newJson*,*oldJson*){  *console*.log(newJson);//等价于oldJson  *console*.log(oldJson);  },{deep:true}); //深度监视 |

## vm.$set

## vm.$delete

# 实例方法/事件

## vm.$on

在构造器的外部增加构造器内部的事件，可以在构造器外部调用内部的数据

|  |
| --- |
| <div id="app">  <p>{{num}}</p>  <button @click="add">add</button>  </div>  <p><button onclick="reduce()">reduce</button></p>  <p><button onclick="reduceOnce()">reduceOnce</button></p>  <p><button onclick="off()">off</button></p>  *var* app=new Vue({  el:'#app',  data:{num:1},  methods:{  add:*function*(){  this.num++;  }  }  })  //实例事件  app.$on('reduce',*function*(){  *console*.log('执行了reduce()');  this.num--;  });  //只使用一次的实例方法  app.$once('reduceOnce',*function*(){  *console*.log('只执行一次的方法');  this.num--;  });  //关闭事件  *function* off(){  app.$off('reduce');  }  //外部调用内部事件  *function* reduce(){  app.$emit('reduce');  }  *function* reduceOnce(){  app.$emit('reduceOnce');  } |

## vm.$once

## vm.$off

## vm.$emit

# 实例方法/生命周期

## vm.$mount

|  |
| --- |
| new Vue({  data:{msg:'vue'}  }).$mount('#app'); // 手动挂载 |

|  |
| --- |
| 使用构造器后挂载  <div id="app"></div>  <p><button onclick="destroy()">卸载</button></p>  <p><button onclick="reload()">刷新</button></p>  <p><button onclick="tick()">修改</button></p>  *var* jspang = Vue.extend({  template:`<p>{{message}}</p>`,  data:*function*(){  return {  message:'Hello ,I am JSPang'  }  },  destroyed:*function*(){  *console*.log('10-destroyed 销毁之后');  },  mounted:*function*(){  *console*.log("4-mounted 被创建");  },  updated:*function*(){  *console*.log("6-updated 被更新后");  }  })  *var* vm = new jspang().$mount("#app"); // 会打印4-mounted 被创建"，**说明在mounted钩子里挂载的**  *function* destroy(){  vm.$destroy(); // destroyed**钩子执行**  }  *function* reload(){  vm.$forceUpdate(); // 更新/刷新方法， updated**钩子执行**  }  *function* tick(){  // 修改数据  vm.message="update message info ";  // DOM 还没有更新  vm.$nextTick(*function*(){ // 是个回调，dom更新后执行  // DOM 现在更新了  // `this` 绑定到当前实例  *console*.log('message更新完后我被调用了');  })  } |

## vm.$forceUpdate

## vm.$nextTick

## vm.$destroy

|  |
| --- |
| vm.$destroy(); //解绑 |

# transition vue1.0

|  |
| --- |
| #div{  *width*:100px;  *height*:100px;  *background*: red;   }  .fade-transition{*transition*: 1s all ease;}  .fade-enter{*opacity*: 0;} /\*进入 \*/  .fade-leave{ /\*离开\*/  *opacity*: 0;  *transform*: translateX(200px);  }  <input type="button" value="按钮" @click="toggle">  <div id="div" v-show="bSign" transition="fade"></div>  toggle(){      this.bSign=!this.bSign;  }  ===========================================  <link rel="stylesheet" href="bower\_components/animate.css/animate.css">  #div{  *width*:100px;  *height*:100px;  *background*: red;  }  <input type="button" value="按钮" @click="toggle">  <div id="div" class="animated" v-show="bSign" transition="bounce"></div>  data:{bSign:true},  methods:{      toggle(){this.bSign=!this.bSign;}  },  transitions:{ //定义所有动画名称      bounce:{             enterClass:'zoomInLeft',             leaveClass:'zoomOutRight'       }  } |

# transition vue2.0

|  |
| --- |
| .fade-enter{} //初始状态  .fade-enter-active{} //变化成什么样 -> 当元素出来(显示)  .fade-leave{}  .fade-leave-active{} //变成成什么样 -> 当元素离开(消失)  //点击显示和隐藏  p{  *width*:300px;  *height*:300px;  *background*: red;  }  .fade-enter-active, .fade-leave-active{  *transition*: 1s all ease;  }  .fade-enter-active{  *opacity*:1;  *width*:300px;  *height*:300px;  }  .fade-leave-active{  *opacity*:0;  *width*:100px;  *height*:100px;  }  .fade-enter,.fade-leave{  *opacity*:0;  *width*:100px;  *height*:100px;  }  <input type="button" value="点击显示隐藏" @click="show=!show">  <transition name="fade">  <p v-show="show"></p>  </transition> |

# transition配合animate

|  |
| --- |
| p{  *width*:100px;  *height*:100px;  *background*: red;  *margin*:0 auto;  }  <link rel="stylesheet" href="animate.css">  <input type="button" value="点击显示隐藏" @click="show=!show">  <transition enter-active-class="animated zoomInLeft"  leave-active-class="animated zoomOutRight">  <p v-show="show"></p>  </transition>  ==================================================  <input type="button" value="点击显示隐藏" @click="show=!show">  <transition name="transition">  <p v-show="show" class="animated"></p>  </transition>  data:{  show:false  },  transitions:{ //错的  enterActiveClass:'zoomInLeft',  leaveActiveClass:'zoomOutRight'  } |

# transition vue2.0-多个元素

|  |
| --- |
| p{  *width*:100px;  *height*:100px;  *background*: red;  *margin*:10px auto;  }  <link rel="stylesheet" href="animate.css">  <input type="text" v-model="show">  <transition-group enter-active-class="zoomInLeft"  leave-active-class="zoomOutRight">  <p v-show="show" class="animated" v-for="(val,index) in lists" :key="index">  {{val}}  </p>  </transition-group>  data:{  show:'',  list:['apple','banana','orange','pear']  },  computed:{  lists:*function*(){  *var* arr=[];  this.list.forEach(*function*(*val*){  if(val.indexOf(this.show)!=-1){  arr.push(val);  }  }.bind(this));  return arr;  }  } |

# transition vue2.0相关函数

|  |
| --- |
| p{  *width*:300px;  *height*:300px;  *background*: red;  }  .fade-enter-active, .fade-leave-active{  *transition*: 1s all ease;  }  .fade-enter-active{  *opacity*:1;  *width*:300px;  *height*:300px;  }  .fade-leave-active{  *opacity*:0;  *width*:100px;  *height*:100px;  }  .fade-enter,.fade-leave{  *opacity*:0;  *width*:100px;  *height*:100px;  }  <input type="button" value="点击显示隐藏" @click="show=!show">  <transition name="fade"  @before-enter="beforeEnter"  @enter="enter"  @after-enter="afterEnter"  @before-leave="beforeLeave"  @leave="leave"  @after-leave="afterLeave">  <p v-show="show"></p>  </transition>  data:{  show:false  },  methods:{  beforeEnter(*el*){  *console*.log('动画enter之前');  },  enter(*el*){  *console*.log('动画enter进入');  },  afterEnter(*el*){  *console*.log('动画进入之后');  el.style.background='blue';  },  beforeLeave(*el*){  *console*.log('动画leave之前');  },  leave(*el*){  *console*.log('动画leave');  },  afterLeave(*el*){  *console*.log('动画leave之后');  el.style.background='red';  }  } |

# 路由配合运动

|  |
| --- |
| .router-link-active{  *font-size*: 20px;  *color*:#f60;  }  <transition enter-active-class="animated bounceInLeft"  leave-active-class="animated bounceOutRight">  <router-view></router-view>  </transition>  router.push({path:'home'});  router.replace({path:'user'}); |

# 父子组件

**vm.$dispatch(事件名,数据) 子级向父级发送数据**

**vm.$broadcast(事件名,数据) 父级向子级广播数据**

**配合: event:{}，但在vue2.0里面已经，报废了**

**vue默认情况下，子组件没法访问父组件数据，子组件获取父组件data**

|  |
| --- |
| <aaa></aaa>  new Vue({  **el**:"#app",  components:{  'aaa':{  data(){  return{ msg:'父组件数据', info:123 }  },  // 给子组件绑定一个属性，指向父组件的msg，在子组件通过props属性，  template:'<h2>aaa组件{{msg}}</h2><bbb :mmm="msg" :my-msg="info"></bbb>',  **components**:{  'bbb':{  // props:{ 'mmm':String, 'myMsg':Number },  props:['mmm','myMsg'],  template:'<h3>bbb组件-->{{mmm}}哈{{myMsg}}</h3>'  }  }  }  }  }) |

**父组件获取子组件data**

|  |
| --- |
| ***子组件把自己的数据，发送到父级,在父组件通过 @child-msg事件名接收***  ***vm.$emit(事件名,数据);***  <div id="box">  <aaa></aaa> </div>  <template id="aaa">  <span>我是父级 -> {{msg}}</span>  <bbb @child-msg="get"></bbb> </template> <template id="bbb">  <h3>子组件-</h3>  <input type="button" value="send" @click="send"> </template>  var vm=new Vue({  **el**:'#box',  components:{  'aaa':{  data(){  return { msg:'父组件的数据'}  },  template:'#aaa',  methods:{  get(**msg**){ this.**msg**=**msg**; }  },  components:{  'bbb':{  data(){  return {a:'子组件的数据'}  },  template:'#bbb',  methods:{  send(){  this.$emit('child-msg',this.**a**);//把子组件的a数据发送给父组件  }  }  }  }  }  } }); |

# vue2.0父子组件

|  |
| --- |
| vm.$emit() vm.$on();  vue1.0，子组件加sync可以同步更改父组件信息， vue2.0，不管是否有sync,都不允许子组件直接改父级的数据，做赋值操作  vue2.0想更改父组件数据  a). 父组件每次传一个对象给子组件, 对象之间引用 √  b). 只是不报错, mounted中转  <script src="vue1.0.js"></script>  <div id="box">  父级: ->{{a}}  <br>  <child-com :msg.sync="a"></child-com> </div>  <template id="child">  <div>  <span>我是子组件</span>  <input type="button" value="按钮" @click="change">  <strong>{{msg}}</strong>  </div>  </template>  new Vue({  **el**:'#box',  **data**:{ **a**:'我是父组件数据'},  components:{  'child-com':{  **props**:['msg'],  **template**:'#child',  methods:{  change(){  this.**msg**='被更改了'  }  }  }  } }); |

vue2.0如下同步父组件数据: a). 父组件每次传一个对象给子组件, 对象之间引用 √

|  |
| --- |
| <div id="box">  父级: ->{{giveData.a}}  <br>  <child-com :msg="giveData"></child-com> </div>  <template id="child">  <div>  <span>我是子组件</span>  <input type="button" value="按钮" @click="change">  <strong>{{msg.a}}</strong>  </div>  </template>  new Vue({  **el**:'#box',  **data**:{  **giveData**:{**a**:'我是父组件数据'}  },  components:{  'child-com':{  **props**:['msg'],  **template**:'#child',  methods:{  change(){  this.**msg**.**a**='被改了';  }  }  }  } }); |

vue2.0如下同步父组件数据: **b). 只是不报错, mounted中转**

|  |
| --- |
| <div id="box">  父级: ->{{giveData.a}}  <br>  <child-com :msg="giveData"></child-com> </div> <template id="child">  <div>  <span>我是子组件</span>  <input type="button" value="按钮" @click="change">  <strong>{{b.a}}</strong>  </div> </template>  new Vue({  **el**:'#box',  **data**:{  **giveData**:{ **a**:'我是父组件数据'}  },  components:{  'child-com':{  **props**:['msg'],  **template**:'#child',  data(){ return{ **b**:'' } },  mounted(){ this.**b**=this.**msg**;},  methods:{  **change**(){  this.**b**.**a**='被改了'; // 等价于this.msg.a  }  }  }  } });  ***如下是否加sync都不能同步***  <div id="box">  父级: ->{{a}}  <br>  <child-com :msg.sync="a"></child-com> </div> <template id="child">  <div>  <span>我是子组件</span>  <input type="button" value="按钮" @click="change">  <strong>{{b}}</strong> <!--b子组件的数据-->  </div>  </template>  new Vue({  **el**:'#box',  **data**:{ **a**:'我是父组件数据'},  components:{  'child-com':{  **props**:['msg'],  **template**:'#child',  data(){ return { **b**:''} },  mounted(){ this.**b**=this.**msg**;},  methods:{  **change**(){ this.**b**='被改了';}  }  }  } }); |

# 单一事件中心管理组件通信

|  |
| --- |
| Event.$emit(事件名称, 数据)  Event.$on(事件名称,function(data){ //data }.bind(this));  <div id="box">  <com-a></com-a>  <com-b></com-b>  <com-c></com-c>  </div>  //准备一个空的实例对象  *var* Event=new Vue();  *var* A={  template:`  <div>  <span>我是A组件</span> -> {{a}}  <input type="button" value="把A数据给C" @click="send">  </div>`,  methods:{  send(){  Event.$emit('a-msg',this.a); // 把数据给别人  }  },  data(){  return {a:'我是a数据'}  }  };  *var* B={  template:`  <div>  <span>我是B组件</span> -> {{a}}  <input type="button" value="把B数据给C" @click="send">  </div>`,  methods:{  send(){  Event.$emit('b-msg',this.a);  }  },  data(){  return {  a:'我是b数据'  }  }  };  *var* C={  template:`  <div>  <h3>我是C组件</h3>  <span>接收过来的A的数据为: {{a}}</span>  <br>  <span>接收过来的B的数据为: {{b}}</span>  </div>`,  data(){  return {  a:'',  b:''  }  },  mounted(){  //var \_this=this;  //接收A组件的数据  Event.$on('a-msg',*function*(*a*){  this.a=a;  }.bind(this));  //接收B组件的数据  Event.$on('b-msg',*function*(*a*){  this.b=a;  }.bind(this));  }  }; |

# vue全局组件use使用

|  |
| --- |
| Loading.vue 写组件代码  index.js  import LoadingComponent from './Loading.vue'  *const* Loading = {    install: *function*(*Vue*) {      Vue.component('Loading', LoadingComponent)    }  };  export default Loading  main.js 引入  import Loading from './components/loading'  Vue.use(Loading)  App.vue 使用  <Loading></Loading> |

# element-ui

|  |
| --- |
| element-ui.js 按需引入  // 按钮和单选....  import {Button,Radio,DatePicker,Rate,Pagination} from 'element-ui'  Vue.use(Button);  Vue.use(Radio);  // tabs  import {TableColumn,Table,Switch,Badge,TabPane,Tabs} from 'element-ui'  Vue.use(Badge);  Vue.use(Table);  main.js // 直接引入  import './element-ui.js' |

# vue-router

<https://jspang.com/post/vue-router.html>

|  |
| --- |
| router路由  routes 路线  this.$route.params  this.$router.go()  <router-link :to='/home'></router-link>  第一种传递参数 name  routes: [  {  path: '/',  name: 'HelloWorld',  component: HelloWorld  },  {  // 有子路由的在子路由里传递name  path:'/hi',  component:Hi,  children: [  {path:'/', name: hi,component: Hi1},  {path:'/hi1', name: 'hi1',component: Hi1},  ]  }  ]  接收{{$route.name}}  第二种  <router-link :to="{name:'hi1',params:{username:'jspang', id:10}}">Hi页面1</router-link>  {{$route.params.username}}  {$route.params.id}} |

# vuex

<https://jspang.com/post/vuex.html>

|  |
| --- |
| types.js  export *const* INCREMENT = 'INCREMENT';  export *const* DECREMENT = 'DECREMENT';  getters.js  export default {    count: (*state*) *=>* {      return state.count;    },    getOdd: (*state*) *=>* {      return state.count % 2 == 0 ? '偶数' : '奇数'    }  }  mutations.js  import {INCREMENT,DECREMENT} from './types'  import getters from './getters'  *const* state = {    count: 20  };  *const* mutations = { //处理状态(数据)变化    [INCREMENT](*state*) {      state.count++;    },    [DECREMENT](*state*) {      state.count--;    }  };  export default {    state,    mutations,    getters  }  actions.js  import \* as types from './types'  export default {    increment: ({ // 处理你要干什么，异步请求，判断，流程控制      commit    }) *=>* {      commit(types.INCREMENT);    },    decrement: ({      commit    }) *=>* {      commit(types.DECREMENT);    },    clickOdd: ({      commit,      state    }) *=>* {      if (state.mutations.count % 2 == 0) {        commit(types.INCREMENT);      }    },    clickAsync: ({      commit    }) *=>* {      new Promise((*resolve*) *=>* {        setTimeout(*function*() {          commit(types.INCREMENT);        }, 1000);      })    }  }  **index.js**  import Vue from 'vue'  import Vuex from 'vuex'  import mutations from './mutations'  import actions from './actions'  Vue.use(Vuex);  export default new Vuex.Store({    modules:{      mutations    },    actions  });  main.js  import Vue from 'vue'  import App from './App.vue'  import store from './store/'  new Vue({    store,    el: '#app',    render: *h* *=>* h(App)  })  App.vue  <h3>welcome vuex-demo</h3>  <input type="button" value="增加" @click="increment">  <input type="button" value="减少" @click="decrement">  <input type="button" value="偶数才能点击+" @click="clickOdd">  <input type="button" value="点击异步" @click="clickAsync">  <div>  现在数字为: {{count}}, 它现在是 {{getOdd}}  </div>  import {mapGetters, mapActions} from 'vuex'  export default{  computed:mapGetters([  'count',  'getOdd'  ]),  methods:mapActions([  'increment',  'decrement',  'clickOdd',  'clickAsync'  ])  } |

# vue-router路由(过时)

根据不同url地址，出现不同效果

|  |
| --- |
| <script src="js/vue.js"></script> <script src="js/vue-router.js"></script>  <div id="box">  <ul>  <!--跳转链接，跳转的页面路径-->  <li><a v-link="{path:'/home'}">主页</a></li>  <li><a v-link="{path:'/news'}">新闻</a></li>  </ul>  <div><router-view></router-view></div> <!--展示内容--> </div>  var App=Vue.extend(); //1. 准备一个根组件 var Home=Vue.extend({ //2. Home News组件都准备  **template**:'<h3>我是主页</h3>' });  var News=Vue.extend({  **template**:'<h3>我是新闻</h3>' }); var router=new **VueRouter**();//3. 准备路由 router.map({ //4. 关联  **'home'**:{  **component**:Home // 要跳转的组件名  },  **'news'**:{  **component**:News  } });  router.start(App,'#box'); //5. 启动路由  //6. 跳转(先启动路由再跳转） router.redirect({  **'/'**:'home' ***//刚打开页面有效 ,'/ccc':'home'打开ccc才跳转，可以多个*** }); |

# vue-router路由嵌套(过时)

|  |
| --- |
| .v-link-active{ /\*v-link自动添加\*/  font-size: 20px;  color: #f60; }  <div id="box">  <ul>  <li><a v-link="{path:'/home'}">主页</a></li>  <li><a v-link="{path:'/news'}">新闻</a></li>  </ul>  <div><router-view></router-view></div> </div>  <template id="home">  <h3>我是主页</h3>  <div>  <a v-link="{path:'/home/login'}">登录</a>  <a v-link="{path:'/home/reg'}">注册</a>  </div>  <div><router-view></router-view></div> </template> <template id="news">  <h3>我是新闻</h3>  <div>  <a v-link="{path:'/news/detail/001'}">新闻001</a>  <a v-link="{path:'/news/detail/002'}">新闻002</a>  </div>  <router-view></router-view> </template> <template id="detail">  {{$route.params | json}} <!--当前参数 001/002-->  <br>  {{$route.path}} <!--当前路径 /news/detail/001 或 002 -->  <br>  {{$route.query | json}} <!--数据 url中vue.html#/news/detail/001?a=1&b=2 (自己写上)  =>{“a”:”1”,”b”:”2”}--> </template>  <script>  //1. 准备一个根组件  var App=Vue.extend();  //2. Home News组件都准备  var Home=Vue.extend({  **template**:'#home'  });  var News=Vue.extend({  **template**:'#news'  });  var Detail=Vue.extend({  **template**:'#detail'  });   //3. 准备路由  var router=new **VueRouter**();  //4. 关联  router.map({  **'home'**:{  **component**:Home,  **subRoutes**:{  **'login'**:{  **component**:{  // { "name": "zns" }  **template**:'<strong>我是登录信息{{$route.params | json}} </strong>' }  },  **'reg'**:{  **component**:{  **template**:'<strong>我是注册信息</strong>'  }  }  }  },  **'news'**:{  **component**:News,  **subRoutes**:{  **'/detail/:id'**:{  **component**:Detail  }  }  }  });   //5. 启动路由  router.start(App,'#box');  //6. 跳转  router.redirect({  **'/'**:'home'  }); </script> |

|  |
| --- |
| <template id="home">  <h3>我是主页</h3>  <div>   ***<a v-link="{path:'/home/login/dmw'}">登录</a>  <a v-link="{path:'/home/reg/lyf'}">注册</a>***  </div>  <div><router-view></router-view></div> </template>  **'home'**:{  **component**:Home,  **subRoutes**:{  **'login/:name'**:{  **component**:{   ***template:'<strong>我是登录信息{{$route.params | json}}</strong> //'{ "name": "dmw" }***  }  },  **'reg'**:{  **component**:{  **template**:'<strong>我是注册信息</strong>'  }  }  } }, |

# vue-resource交互(过时)

|  |
| --- |
| <script src="vue.js"></script> <script src="vue-resource.js"></script>  methods:{  get:function(){  this.**$http**.get('get.php',{  **a**:1,  **b**:2  }).then(function(res){  alert(res.**data**);  },function(res){  alert(res.**status**);  });  } }  <?php  $a=$\_GET['a'];  $b=$\_GET['b'];  echo $a+$b;  ?>  methods:{  post:function(){  this.**$http**.**post**('post.php',{  **a**:1,  **b**:20  },{  **emulateJSON**:true  }).then(function(res){  alert(res.**data**);  },function(res){  alert(res.**status**);  });  } }  <?php  $a=$\_POST['a'];  $b=$\_POST['b'];  echo $a-$b;  ?>  ***https://sug.so.360.cn/suggest?callback=suggest\_so&word=a***  methods:{  **jsonp**:function(){  this.**$http**.**jsonp**('https://sug.so.360.cn/suggest',{  **word**:'a'  }).then(function(res){  alert(res.**data**.s); // 返回数据里的s属性值  },function(res){  alert(res.**status**);  });  } }  ***https://sp0.baidu.com/5a1Fazu8AA54nxGko9WTAnF6hhy/su?wd=a&cb=jshow***  methods:{  jsonp:function(){  this.**$http**.**jsonp**('https://sp0.baidu.com/5a1Fazu8AA54nxGko9WTAnF6hhy/su',{  **wd**:'a'  },{  **jsonp**:'cb' //callback名字，默认名字就是"callback"  }).then(function(res){  alert(res.**data**.s);  },function(res){  alert(res.**status**);  });  } }  this.**$http**({  **url**:地址  **data**:给后台提交数据,  **method**:'get'/post/jsonp  **jsonp**:'cb' //cbName }); |

技术胖

|  |
| --- |
| <script type="text/javascript" src="../assets/js/vue.js"></script>  <script src="https://cdn.jsdelivr.net/vue.resource/1.2.1/vue-resource.min.js"></script>  {{message}}  *var* app=new Vue({  el:'#app',  data:{  message:'hello Vue!'  },  beforeMount:*function*(){  *console*.log('Vue挂载之前.');  //get 方式请求  this.$http.get('/example/1.txt').then((*response*)*=>*{  //success callback  *console*.log(response);  this.message=response.body;  },(*response*)*=>*{  //error callback  *console*.log('ajax error!');  });  //Post 方式请求  *var* formData =new FormData();  formData.append('a','1');  this.$http.post('/example/1.txt',formData).then((*response*)*=>*{  //success callback  *console*.log(response);  this.message=response.body;  },(*response*)*=>*{  //error callback  *console*.log('ajax error!');  });  }  }) |

# vue里使用jquery

|  |
| --- |
| <script type="text/javascript" src="./jquery-3.1.1.min.js"></script>  *var* app=new Vue({  el:'#app',  data:{  message:'hello Vue!'  },  //在Vue中使用jQuery  mounted:*function*(){  // 挂载了之后才能获取到元素  $('#app').html('我是jQuery!');  },  methods:{  add:*function*(){  *console*.log("调用了Add方法");  }  }  })  app.add(); |

# 百度下拉列表

|  |
| --- |
| .gray{*background*: #ccc;}  <input type="text"  v-model="t1"  @keyup="get($event)"  @keydown.down="changeDown()"  @keydown.up.prevent="changeUp()">  <ul>  <li v-for="value in myData" :class="{gray:$index==now}">{{value}} </li>  </ul>  <p v-show="myData.length==0">暂无数据...</p>  <script src="vue.js"></script>  <script src="vue-resource.js"></script>  new Vue({  el:'body',  data:{  myData:[],  t1:'',  now:-1  },  methods:{  get:*function*(*ev*){  if(ev.keyCode==38 || ev.keyCode==40)return; // 上下键不触发请求  if(ev.keyCode==13){ // open() 方法用于打开一个新的浏览器窗口或查找一个已命名的窗口  window.open('https://www.baidu.com/s?wd='+this.t1);  this.t1='';  }  this.$http.jsonp('https://sp0.baidu.com/5a1Fazu8AA54nxGko9WTAnF6hhy/su',{  wd:this.t1  },{  jsonp:'cb'  }).then(*function*(*res*){  this.myData=res.data.s;  },*function*(){  });  },  changeDown:*function*(){  this.now++;  if(this.now==this.myData.length)this.now=-1;  this.t1=this.myData[this.now];  },  changeUp:*function*(){  this.now--;  if(this.now==-2)this.now=this.myData.length-1;  this.t1=this.myData[this.now];  }  }  }); |

# 简易留言板

|  |
| --- |
| th,td {*width*: 25%;}  <div class="container" id="box">  <form role="form">  <div class="form-group">  <label for="username">用户名:</label>  <input type="text" id="username" class="form-control" placeholder="输入用户名"  v-model="username">  </div>  <div class="form-group">  <label for="age">年　龄:</label>  <input type="text" id="age" class="form-control" placeholder="输入年龄" v-model="age">  </div>  <div class="form-group">  <input type="button" value="添加" class="btn btn-primary" v-on:click="add()">  <input type="reset" value="重置" class="btn btn-danger">  </div>  </form>  <hr>  <table class="table table-bordered table-hover">  <caption class="h3 text-info">用户信息表</caption>  <tr class="text-danger">  <th class="text-center">序号</th>  <th class="text-center">名字</th>  <th class="text-center">年龄</th>  <th class="text-center">操作</th>  </tr>  <tr class="text-center" v-for="item in myData">  <td>{{$index+1}}</td>  <td>{{item.name}}</td>  <td>{{item.age}}</td>  <td>  <button class="btn btn-primary btn-sm" data-toggle="modal" data-target="#layer"  v-on:click="nowIndex=$index">删除</button>  </td>  </tr>  <tr v-show="myData.length!=0">  <td colspan="4" class="text-right">  <button class="btn btn-danger btn-sm" v-on:click="nowIndex=-2" data-toggle="modal"  data-target="#layer" >删除全部</button>  </td>  </tr>  <tr v-show="myData.length==0">  <td colspan="4" class="text-center text-muted">  <p>暂无数据....</p>  </td>  </tr>  </table>  <!--模态框 弹出框-->  <div role="dialog" class="modal fade bs-example-modal-sm" id="layer">  <div class="modal-dialog">  <div class="modal-content">  <div class="modal-header">  <button type="button" class="close" data-dismiss="modal">  <span>&times;</span>  </button>  <h4 class="modal-title">确认删除么?</h4>  </div>  <div class="modal-body text-right">  <button data-dismiss="modal" class="btn btn-primary btn-sm">取消</button>  <button data-dismiss="modal" class="btn btn-danger btn-sm"  v-on:click="deleteMsg(nowIndex)">确认</button>  </div>  </div>  </div>  </div>  </div>  <link rel="stylesheet" href="bootstrap.min.css"/>  <script src="jquery-1.11.1.js"></script>  <script src="bootstrap.js"></script>  <script src="node\_modules/vue/dist/vue.js"></script>  new Vue({  el:'#box',  data:{  myData:[],  username:'',  age:'',  nowIndex:-100  },  methods:{  add:*function*(){  this.myData.push({  name:this.username,  age:this.age  });  this.username='';  this.age='';  },  deleteMsg:*function*(*n*){  if(n==-2){  this.myData=[];  }else{  this.myData.splice(n,1);  }  }  }  }); |