vue常用指令

**基础**

1.v-model 一般用于表单元素 双向数据绑定

(个人抽象的认为 v-model=表单.value)

{{msg}} 数据更新模版变化

{{\*msg}}数据只更新一次，以后不再跟新

vue2.0 <p v-once>{{message}}</p>

{{{msg}}} 可转义输入的html标签

修饰符：

.lazy

|  |
| --- |
| <!-- 在 "change" 而不是 "input" 事件中更新 -->  <input v-model.lazy="msg" > |

.number

自动将用户的输入值转为 Number 类型（如果原值的转换结果为 NaN 则返回原值）

.trim

自动过滤用户输入的首尾空格

2.new Vue({

el: 选择器 class id label

data: {

名称(msg)：字符／数字／数组／json

},

methods{

函数：function(){

}

}

})

3.循环：(见demo6)

v-for = “name in arr”

<li v-for="(value,index) in arr">

{{value}} {{index}}

</li>

<li v-for="(value,key) in json">

{{value}} {{key}}

</li>

4.事件

v-on:click="函数 "

显示隐藏：v-show:”true/false”

5. v-text 更新元素的 textContent

6.过滤器 –- 过滤模版数据 （2.0以废弃，需要自己定义）

Vue 2.x 过滤器只能用于双花括号插值(mustache interpolation)和 v-bind 表达式中

系统提供的一些过滤器：

| uppercase 小写转大写

| lowercase 大写转小写

| capitalize 首字母大写

|

多过滤器 {{msg | filterA | filterB}}

7.交互

$http (ajax)

如果vue想做交互，需要引入：vue-resource

8.全局配置

Vue.config：是一个对象，包含 Vue 的全局配置

属性：

1.Vue.config.silent = true

silent:是否取消Vue 所有的日志与警告。

2.

9. 半角句号 . 指明的特殊后缀，用于指出一个指令应该以特殊方式绑定

10.v-bind : click 缩写 ：click ---针对属性

v-on:click 缩写 @click ---针对事件

11. **计算属性（computed）是基于它们的依赖进行缓存的**。计算属性只有在它的相关依赖发生改变时才会重新求值

12．条件的判断

data{ a=1,}

<p v-if="a==1">ok</p>

<p v-else>no</p>

13. Vue 包含一组观察数组的变异方法，所以它们也将会触发视图更新。这些方法如下：

* push() 添加数组元素
* pop() 弹出数组元素

shift() 把数组的第一个元素从其中删除，并返回第一个元素的值。

unshift() 向数组的开头添加一个或更多元素，并返回新的长度。

splice() 向/从数组中添加/删除项目，然后返回被删除的项目。

* sort()
* reverse() 反转

利用索引直接改变一个选项：

Vue.set(example1.items, indexOfItem, newValue)

或example1.items.splice(indexOfItem, 1, newValue)

修改数组长度：

|  |
| --- |
| example1.items.splice(newLength) |

14.重塑数组：

filter(), concat() 和 slice() 返回新数组

filter() 方法创建一个新的数组，新数组中的元素是通过检查指定数组中符合条件的所有元素。

slice() 方法可从已有的数组中返回选定的元素。

concat() 方法用于连接两个或多个数组。返回被连接数组的一个副本。

15.  Vue.js 为 v-on 提供了 **事件修饰符**

.stop

<!-- 阻止单击事件冒泡 -->

<a v-on:click.stop="doThis"></a>

.prevent

<!-- 提交事件不再重载页面 -->

<form v-on:submit.prevent="onSubmit"></form>

.capture

<!-- 添加事件侦听器时使用事件捕获模式 -->

<div v-on:click.capture="doThis">...</div>

.self

<!-- 只当事件在该元素本身（比如不是子元素）触发时触发回调 -->

<div v-on:click.self="doThat">...</div>

.once

<!-- 点击事件将只会触发一次 -->

<a v-on:click.once="doThis"></a>

16. 使用修饰符时，顺序很重要；相应的代码会以同样的顺序产生。因此，用 @click.prevent.self 会阻止**所有的点击**，而 @click.self.prevent 只会阻止元素上的点击。

17.键盘监听---键值修饰符

Vue 允许为 v-on 在监听键盘事件时添加关键修饰符

v-on:keyup.

* .enter
* .tab
* .delete (捕获 “删除” 和 “退格” 键)
* .esc
* .space
* .up
* .down
* .left
* .right

18.什么是组件(注：组件命名时不能用驼峰式)

组件 (Component) 是 Vue.js 最强大的功能之一。组件可以扩展 HTML 元素，封装可重用的代码。在较高层面上，组件是自定义元素，Vue.js 的编译器为它添加特殊功能。在有些情况下，组件也可以是原生 HTML 元素的形式，以 is 特性扩展。（要确保在初始化根实例**之前**注册了组件）

注册一个全局组件；

Vue.component('my-component', {

// 选项

})

//定义

var MyComponent = Vue.extend({

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

})

// 注册

Vue.component('my-component', MyComponent)

局部注册：

通过使用组件实例选项注册，可以使组件仅在另一个实例/组件的作用域中可用

19. 不能传递数据给组件，因为组件的作用域是独立的。为了传递数据给组件，应当使用props

20. 变化检测问题：vue.js不能检测到对象属性的添加或删除，属性必须在data上才能让vue.js转换它为getter/setter模式，才能有响应。

21. <ul>，<ol>，<table>，<select> 限制了能被它包裹的自定义组件中使用

例如：自定义组件 <my-row>

错误：<table>

<my-row>...</my-row>

</table>

正确：<table>

<tr is="my-row"></tr>

</table>

22. 父组件通过 **props** 向下传递数据给子组件，子组件通过 **events** 给父组件发送消息（见demo7-2）

Vue.component("todo-item",{

props:['todo'],

template:'<li>{{ todo }}</li>'

})

var v = new Vue({

el:'#box',

data:{

array:['a','b','c','d','e']

}

})

<todo-item v-for="item in array" v-bind:todo="item" v-bind:key="item.index"></todo-item>

如果你想要用一个对象作为 props 传递所有的属性，你可以使用不带任何参数的 v-bind

例：若todo是一个对象可 v-bind:todo

如果想传递一个实际的 number，需要使用 v-bind，从而让它的值被当作 JavaScript 表达式计算

例：若todo要接收一个对象， v-bind:todo=”1”

组件的单项数据流：

prop 是单向绑定的：当父组件的属性变化时，将传导给子组件，但是不会反过来。这是为了防止子组件无意修改了父组件的状态——这会让应用的数据流难以理解。

另外，每次父组件更新时，子组件的所有 prop 都会更新为最新值。这意味着你**不应该**在子组件内部改变 prop。如果你这么做了，Vue 会在控制台给出警告。

注意在 JavaScript 中对象和数组是引用类型，指向同一个内存空间，如果 prop 是一个对象或数组，在子组件内部改变它**会影响**父组件的状态。

注意：props可以指定验证规格，传入数据不符合规格，Vue会报错。

非prop特性：

当它要和一个 Bootstrap 插件相互操作时，需要在这个第三方组件的 input 上添加 data-3d-date-picker 特性，这时可以把属性直接添加到组件上

例：<todo data-3d-date-picker=’true’></todo>

合并已有属性：

若要在组件原有属性上添加额外新属性，这两个特性的值都会做合并 (merge) 操作，让最终生成的值

23. HTML 特性是不区分大小写的

24．组件用prop传递时加加冒号的(v-bind:的简写)，说明=后面的是一个变量或者表达式，没加冒号的=后面就是对应的字符串字面量

25.$emit(event,[…args]) 触发当前实例上的事件。附加参数都会传给监听器回调

$on(event,callback) 监听当前实例上的自定义事件。事件可以由vm.$emit触发。回调函数会接收所有传入事件触发函数的额外参数。

26. 组件作用域：父组件模板的内容在父组件作用域内编译；子组件模板的内容在子组件作用域内编译。

27.slot插槽：当子组件模板只有一个没有属性的 slot 时，父组件整个内容片段将插入到 slot 所在的 DOM 位置，并替换掉 slot 标签本身。假如父组件需要在子组件内放一些DOM，那么这些DOM是显示、不显示、在哪个地方显示、如何显示，就是slot分发负责的活。

例如：template:'<button><slot name="first"></slot>这是slot<slot name="second"></slot></button>'

<child>

<p slot="first">ppp1</p>

<p slot="second">ppp2</p>

</child>

28.异步组件：Vue.js 允许将组件定义为一个工厂函数，动态地解析组件的定义

Vue.component('async-example', function (resolve, reject) {

setTimeout(function () {

// Pass the component definition to the resolve callback

resolve({

template: '<div>I am async!</div>'

})

}, 1000)

})

工厂函数接收一个 resolve 回调，在收到从服务器下载的组件定义时调用。也可以调用 reject(reason) 指示加载失败

29. Vue 会在初始化实例时对属性执行 getter/setter 转化过程，所以属性必须在 data 对象上存在才能让 Vue 转换它，这样才能让它是响应的

30.单元素／组件的过渡

Vue 提供了 transition 的封装组件，可以给任何元素和组件添加 entering/leaving 过渡

当插入或删除包含在 transition 组件中的元素时，Vue 将会做以下处理：

1. 自动嗅探目标元素是否应用了 CSS 过渡或动画，如果是，在恰当的时机添加/删除 CSS 类名。
2. 如果过渡组件提供了 [**JavaScript 钩子函数**](https://cn.vuejs.org/v2/guide/transitions.html#JavaScript-钩子)，这些钩子函数将在恰当的时机被调用。
3. 如果没有找到 JavaScript 钩子并且也没有检测到 CSS 过渡/动画，DOM 操作（插入/删除）在下一帧中立即执行。(注意：此指浏览器逐帧动画机制，和Vue的 nextTick 概念不同)

31.<transition-group>:会以一个真实元素呈现：默认为一个 <span>。你也可以通过 tag 特性更换为其他元素。

32. <transition-group> 组件还有一个特殊之处。不仅可以进入和离开动画，还可以改变定位。要使用这个新功能只需了解新增的 **v-move 特性**，它会在元素的改变定位的过程中应用。像之前的类名一样，可以通过 name 属性来自定义前缀

32.Watcher 我们能监听到任何数值属性的数值更新

33. 混合 (mixins) 是一种分发 Vue 组件中可复用功能的非常灵活的方式。混合对象可以包含任意组件选项。以组件使用混合对象时，所有混合对象的选项将被混入该组件本身的选项。

34. mixins 选项接受一个混合对象的数组(Array<Object>)。这些混合实例对象可以像正常的实例对象一样包含选项,他们将在 Vue.extend() 里最终选择使用相同的选项合并逻辑合并。

35.Vue实例属性：var vm = new Vue({})

1.vm.$data 用于访问被创建的原始数据对象。访问 vm.a 等价于访问 vm.$data.a

推荐在创建实例之前，就声明所有的根级响应式属性。 （见text.html）

如果需要，可以通过将 vm.$data 传入 JSON.parse(JSON.stringify(...)) 得到深拷贝的原始数据对象。

注意，**不应该对 data 属性使用箭头函数**

2.vm.$props props 可以是数组或对象，用于接收来自父组件的数据。

3.computed的选项／数据(见test2.html) 计算属性将被混入到 Vue 实例中。所有 getter 和 setter 的 this 上下文自动地绑定为 Vue 实例。

4.methods的选项／数据(见test3.html) methods 将被混入到 Vue 实例中。可以直接通过 VM 实例访问这些方法，或者在指令表达式中使用。方法中的 this 自动绑定为 Vue 实例。

5.JavaScript---watch

Vue 实例将会在实例化时调用 $watch()，遍历 watch 对象的每一个属性。

回调函数得到的参数为新值和旧值

36.实例生命周期

每个 Vue 实例在被创建之前都要经过一系列的初始化过程。在这个过程中也会运行一些叫做**生命周期钩子(**是一种事件劫持机制，也就是说它会比你的事件更早进行执行处理。可以简单地把它理解为vue的内置事件，但是这个内置事件是由你去配置的。**)**的函数，给予用户机会在一些特定的场景下添加他们自己的代码。

常用勾子有：created ，mounted，updated，destroyed。。。钩子的 this 指向调用它的 Vue 实例。

37.复杂的数据绑定

{{ number + 1 }}

{{ ok ? 'YES' : 'NO' }}

{{ message.split('').reverse().join('') }}

有个限制就是，每个绑定都只能包含**单个表达式**，所以下面的例子都**不会**生效。

<!-- 这是语句，不是表达式 -->

{{ var a = 1 }}

<!-- 流控制也不会生效，请使用三元表达式 -->

{{ if (ok) { return message } }}

38.Vue指令

指令 (Directives) 是带有 v- 前缀的特殊属性。指令的职责是，当表达式的值改变时，将其产生的连带影响，响应式地作用于 DOM。

一些指令能够接收一个“参数”，在指令名称之后以冒号表示

|  |
| --- |
| <a v-bind:href="url"></a> |

在这里 href 是参数，告知 v-bind 指令将该元素的 href 属性与表达式 url 的值绑定。

39.修饰符

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

<form v-on:submit.prevent="onSubmit"></form>

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

event.preventDefault == .prevent

event.stopPropagation == .stop

40.Vue 计算属性（computed）

将函数包装成一个自定义的属性。

computed:{

reversedMessage:function(){

return this.message.split('').reverse().join('')}

console.log(vm.reversedMessage);

vm.message= "Goodby";

console.log(vm.reversedMessage);

计算属性的缓存 VS 方法：

**计算属性是基于它们的依赖进行缓存的**。计算属性只有在它的相关依赖发生改变时才会重新求值。

计算属性 VS 被观察的属性(见test6-2)

计算属性的setter（见test6-3）计算属性默认只有 getter ，不过在需要时你也可以提供一个 setter

41.Class与Style绑定

因为它们都是属性 ，我们可以用v-bind 处理它们：只需要计算出表达式最终的字符串。 在 v-bind 用于 class 和 style 时，Vue.js 专门增强了它。表达式的结果类型除了字符串之外，还可以是对象或数组。

绑定HTML的Class----对象语法

动态切换class:(见test7)

|  |
| --- |
| 1 <div v-bind:class="{ active: isActive }"></div> |

上面的语法表示 class active 的更新将取决于数据属性 isActive 是否为**[真值](https://developer.mozilla.org/en-US/docs/Glossary/Truthy" \t "_blank)**。

2. 同理也可以直接绑定数据里的一个对象

3. 同理我们也可以在这里绑定返回对象的**[计算属性](https://cn.vuejs.org/v2/guide/computed.html)**。这是一个常用且强大的模式

4. 我们可以把一个数组传给 v-bind:class，以应用一个 class 列表

5. 当你在一个自定义组件上用到 class 属性的时候，这些类将被添加到根元素上面，这个元素上已经存在的类不会被覆盖。

template:'<div class="local"></div>'

<my-component class="active"></my-component>

==》 <div class="local active"></div>

**绑定內联样式(见test8)**

v-bind:style 的对象语法十分直观——看着非常像 CSS，其实它是一个 JavaScript 对象。CSS 属性名可以用驼峰式 (camelCase) 或短横分隔命名

1. 绑定到Vue原始数据
2. 绑定到数据对象

3.数组语法 可以将多个样式对象应用到一个元素上

v-bind：style=”[ ]”

42.条件渲染

v-if 是一个指令，需要将它添加到一个元素上。v-else 元素必须紧跟在 v-if 或者 v-else-if 元素的后面——否则它将不会被识别。v-else-if 必须紧跟在 v-if 或者 v-else-if 元素之后。

把一个 <template> 元素当做包装元素，并在上面使用 v-if。最终的渲染结果不会包含 <template> 元素。

用key管理可复用的元素(见test9)

Vue 会尽可能高效地渲染元素，通常会复用已有元素而不是从头开始渲染。这么做，除了使 Vue 变得非常快之外，还有一些有用的好处。

另一个用于根据条件展示元素的选项是 v-show 指令。不同的是带有 v-show 的元素始终会被渲染并保留在 DOM 中。v-show 是简单地切换元素的 CSS 属性 display 。

注意，v-show 不支持 <template> 语法，也不支持 v-else。

v-if VS v-show

v-if 是“真正的”条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建。

v-if 也是**惰性的**：如果在初始渲染时条件为假，则什么也不做——直到条件第一次变为真时，才会开始渲染条件块。

相比之下，v-show 就简单得多——不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 进行切换。

一般来说，v-if 有更高的切换开销，而 v-show 有更高的初始渲染开销。因此，如果需要非常频繁地切换，则使用 v-show 较好；如果在运行时条件不太可能改变，则使用 v-if 较好。

43.对象更新检测

使用 Vue.set(object, key, value) 方法向嵌套对象添加响应式属性。

Vue.set 全局 或 this.$set

44.v-for

v-for VS v-if

当它们处于同一节点，v-for 的优先级比 v-if 更高，这意味着 v-if 将分别重复运行于每个 v-for 循环中。而如果你的目的是有条件地跳过循环的执行，那么可以将 v-if 置于外层元素

在组件中的v-for

**2.2.0+ 的版本里，当在组件中使用 v-for 时，key 现在是必须的。**

注意：v-for不能自动传递数据到组件里，因为组件有自己独立的作用域。为了传递迭代数据到组件里，我们要用 props

45. 在内联语句处理器中访问原生 DOM 事件。可以用特殊变量 $event 把它传入方法：

**组件创建的规则**

46.DOM模版解析说明

当使用 DOM 作为模板时 (例如，将 el 选项挂载到一个已存在的元素上)，你会受到 HTML 的一些限制，因为 Vue 只有在浏览器解析和标准化 HTML 后才能获取模板内容。尤其像这些元素 <ul>，<ol>，<table>，<select> 限制了能被它包裹的元素，而一些像 <option> 这样的元素只能出现在某些其它元素内部。

例： <table>

<my-row>...</my-row> 无效内容

<tr is=’my-row’>……</tr> tr被is转换成自定义组件

</table>

47 组件中的data必须是函数(见demo10)

例；Vue.component('my-component', {

template: '<span>{{ message }}</span>',

错误 *data: {*

*message: 'hello'*

*}*

转变 data:function(){

return message:’hello’

}

})

48.自定义事件：实现子组件与父组件通信 (demo10)

每个 Vue 实例都实现了**[事件接口 (Events interface)](https://cn.vuejs.org/v2/api/" \l "实例方法-事件)**，即：

* 使用 $on(eventName) 监听事件
* 使用 $emit(eventName) 触发事件

父组件可以在使用子组件的地方直接用 v-on 来监听子组件触发的事件。

例：child-button

子组件：

template: '<button v-on:click="incrementCounter">{{ counter }}</button>',

父组件：

<child-button v-on:increment="incrementTotal"></child-button>

在某个组件的根元素上监听一个原生事件。可以使用 .native 修饰 v-on

|  |
| --- |
| <my-component v-on:click.native="doTheThing"></my-component> |

49.两个组件的通信

可以使用一个空的 Vue 实例作为中央事件总线：

50.使用插槽slot分发内容：

内容分发：混合父组件的内容与子组件自己的模板

父组件模板的内容在父组件作用域内编译；子组件模板的内容在子组件作用域内编译。

错误用法：父组件模板内将一个指令绑定到子组件的属性/方法

<slot> 标签中的任何内容都被视为**备用内容**。备用内容在子组件的作用域内编译，并且只有在宿主元素为空，且没有要插入的内容时才显示备用内容。

slot的作用域插槽：(demo14-2)

在父级中，具有特殊属性 scope 的 <template> 元素必须存在，表示它是作用域插槽的模板。scope 的值对应一个临时变量名，此变量接收从子组件中传递的 props 对象

50 kepp-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">

<!-- 非活动组件将被缓存！ -->

</component>

</keep-alive>

51.组件杂项

编写可复用组件

Vue 组件的 API 来自三部分 - props, events 和 slots ：

* **Props** 允许外部环境传递数据给组件
* **Events** 允许从外部环境在组件内触发副作用
* **Slots** 允许外部环境将额外的内容组合在组件中。

子组件索引

有时需要在 JavaScript 中直接访问子组件。为此可以使用 ref 为子组件指定一个索引 ID。

例： <div id="parent">

<user-profile ref="profile"></user-profile>

</div>

var parent = new Vue({ el: '#parent' })

// 访问子组件

var child = parent.$refs.profile

高级异步组件（工厂函数）

当一个异步组件被作为 vue-router 的路由组件使用时，这些高级选项都是无效的，因为在路由切换前就会提前加载所需要的异步组件。

递归组件

组件在它的模板内可以递归地调用自己，不过，只有当它有 name 选项时才可以

如果你不谨慎，递归组件可能导致死循环：

|  |
| --- |
| name: 'stack-overflow',  template: '<div><stack-overflow></stack-overflow></div>' |

上面组件会导致一个错误“max stack size exceeded”，所以要确保递归调用有终止条件 (比如递归调用时使用 v-if 并让他最终返回 false)。

**可复用性$组合**

1.混合(senior->demo1.html)

混合的使用：

var minxin={ }

new Vue({

mixins:[mixin]

})

混合 (mixins) 是一种分发 Vue 组件中可复用功能的非常灵活的方式。混合对象可以包含任意组件选项。以组件使用混合对象时，所有混合对象的选项将被混入该组件本身的选项。

选项合并 (senior->demo1-2.html)

当组件和混合对象含有同名选项时，这些选项将以恰当的方式混合。比如，同名钩子函数将混合为一个数组，因此都将被调用。另外，混合对象的 钩子将在组件自身钩子 **之前** 调用

值为对象的选项，例如 methods, components 和 directives，将被混合为同一个对象。两个对象键名冲突时，取组件对象的键值对。(senior/demo1-3)

注意：Vue.extend() 也使用同样的策略进行合并。

全局混合(senior/demo1-4.html)

一旦使用全局混合对象，将会影响到 **所有** 之后创建的 Vue 实例。使用恰当时，可以为自定义对象注入处理逻辑

谨慎使用全局混合对象，因为会影响到每个单独创建的 Vue 实例 (包括第三方模板)。

自定义选项合并策略

自定义选项将使用默认策略，即简单地覆盖已有值。如果想让自定义选项以自定义逻辑合并，可以向 Vue.config.optionMergeStrategies 添加一个函数

Vue.config.optionMergeStrategies.myOption = function (toVal, fromVal) {

// return mergedVal

}

2.自定义指令

注册全局指令

Vue.directive(‘focus’,{ })

注册局部指令

directive:{

focus:{ // 指令的定义---

}

}

勾子函数：

指令定义函数提供了几个钩子函数 (可选)：

* bind：只调用一次，指令第一次绑定到元素时调用，用这个钩子函数可以定义一个在绑定时执行一次的初始化动作。
* inserted：被绑定元素插入父节点时调用 (父节点存在即可调用，不必存在于 document 中)。
* update：所在组件的 VNode 更新时调用，**但是可能发生在其孩子的 VNode 更新之前**。指令的值可能发生了改变也可能没有。但是你可以通过比较更新前后的值来忽略不必要的模板更新 (详细的钩子函数参数见下)。
* componentUpdated：所在组件的 VNode **及其孩子的 VNode** 全部更新时调用。
* unbind：只调用一次，指令与元素解绑时调用。

钩子函数的参数 (包括 el，binding，vnode，oldVnode)。（senior／demo2-1）

el: 指令所绑定的元素，可以用来直接操作 DOM 。

binding: 一个对象包涵name,value,oldValue,expression,arg,modifires

vnode: Vue 编译生成的虚拟节点，查阅 [**VNode API**](https://cn.vuejs.org/v2/api/#VNode-接口) 了解更多详情。

oldVnode: 上一个虚拟节点，仅在 update 和 componentUpdated 钩子中可用。

如果需要在钩子之间共享数据，建议通过元素的 [**dataset**](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/dataset) 来进行。

如果指令需要多个值，可以传入一个 JavaScript 对象字面量。

3.**渲染函数 $JSX（生疏待提高）**

通过 level prop 动态生成 heading 标签的组件(senior/demo3.html):

在不同级别的标题中插入锚点元素，我们需要重复地使用 <slot></slot>。

**render**(senior/demo3-1.html) 渲染函数

字符串模板的代替方案，允许你发挥 JavaScript 最大的编程能力。render 函数接收一个 createElement 方法作为第一个参数用来创建 VNode。

vm.$slots.default 用来访问被插槽分发的内容。每个**[具名插槽](https://cn.vuejs.org/v2/guide/components.html" \l "具名插槽)** 有其相应的属性 (例如：slot="foo" 中的内容将会在 vm.$slots.foo 中被找到)default 属性包括了所有没有被包含在具名插槽中的节点。

VNode虚拟节点：“虚拟 DOM”是我们对由 Vue 组件树建立起来的整个 VNode 树的称呼。

VNode必须唯一

createElement（tag，object，[]）生成VNode

参数一：一个 HTML 标签字符串，组件选项对象，或者一个返回值类型为 String/Object 的函数，必要参数

参数二：一个包含模板相关属性的数据对象在 template 中使用这些属性。可选参数。

参数三：子节点 (VNodes)，由 `createElement()` 构建而成，或简单的使用字符串来生成“文本节点”。可选参数。

函数式组件：无状态 (没有 data)，无实例 (没有 this 上下文)。

$slots（）与children 比较

两者功能类似，但slots相对children更有针对性

|  |
| --- |
| <my-functional-component>  <p slot="foo">  first  </p>  <p>second</p>  </my-functional-component> |
|  |
|  |

对于这个组件，children 会给你两个段落标签，而 slots().default 只会传递第二个匿名段落标签，slots().foo 会传递第一个具名段落标签。同时拥有 children 和 slots() ，因此你可以选择让组件通过 slot() 系统分发或者简单的通过 children 接收，让其他组件去处理。

4.插件

插件通常会为 Vue 添加全局功能。

Vue.js 的插件应当有一个公开方法 install 。这个方法的第一个参数是 Vue 构造器，第二个参数是一个可选的选项对象。

通过全局方法 Vue.use() 使用插件

Vue.use 会自动阻止注册相同插件多次，届时只会注册一次该插件。

**Vue – 路由**

**路由的作用：**vue的单页面应用是基于路由和组件的，路由用于设定访问路径，并将路径和组件映射起来。传统的页面应用，是用一些超链接来实现页面切换和跳转的。在vue-router单页面应用中，则是路径之间的切换，也就是组件的切换。（路由用于切换页面）

1.vm.$mount()如果 Vue 实例在实例化时没有收到 el 选项，则它处于“未挂载”状态，没有关联的 DOM 元素。可以使用 vm.$mount() 手动地挂载一个未挂载的实例。

2.从零开始的简单路由(senior/demo4-1)

如果只需要非常简单的路由而不需要引入整个路由库，可以动态渲染一个页面级的组件

3.动态路由匹配(senior/demo4-2.html)

我们经常需要把某种模式匹配到的所有路由，全都映射到同个组件。例如，我们有一个 User 组件，对于所有 ID 各不相同的用户，都要使用这个组件来渲染。那么，我们可以在 vue-router 的路由路径中使用『动态路径参数』（dynamic segment）来达到这个效果

一个『路径参数』使用冒号 : 标记。当匹配到一个路由时，参数值会被设置到 this.$route.params，可以在每个组件内使用。

| **模式** | **匹配路径** | **$route.params** |
| --- | --- | --- |
| /user/:username | /user/evan | { username: 'evan' } |
| /user/:username/post/:post\_id | /user/evan/post/123 | { username: 'evan', post\_id: 123 } |

4.创建路由的一般过程：

1.定义路由组件

const a ={template:’’}

1.5 定义路由

const routes=[

{path:’/..’,template:a}

]

2.创建router实例

const router = new VueRouter({routes})

3.创建和挂载根实例

const vm = new Vue({

router

}).$mount(‘#box’)

5. <router-link> 默认会被渲染成一个 `<a>` 标签

当 <router-link> 对应的路由匹配成功，将自动设置 class 属性值 .router-link-active。

使用router-link组件来导航

<!-- 通过传入 `to` 属性指定链接. -->

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

<router-link to="/foo">Go to Foo</router-link>

使用router-view

<!-- 路由出口 -->

<!-- 路由匹配到的组件将渲染在这里 -->

<router-view></router-view>

6.嵌套路由（senior/demo4-3）

routes:[

path:’/user/:id’,component:User,children[

{ path:’profile’,component:Profile }

]

]

**要注意，以 / 开头的嵌套路径会被当作根路径。 这让你充分的使用嵌套组件而无须设置嵌套的路径。**

基于上面的配置，当你访问 /user/foo 时，User 的出口是不会渲染任何东西，这是因为没有匹配到合适的子路由。如果你想要渲染点什么，可以提供一个 空的 子路由

7.编程式导航

router.push(location)这个方法会向 history 栈添加一个新的记录，所以，当用户点击浏览器后退按钮时，则回到之前的 URL。等同与<router-link :to=’’>

该方法的参数可以是一个字符串路径，或者一个描述地址的对象。

router.replace(location)不会向 history 添加新记录，而是跟它的方法名一样 —— 替换掉当前的 history 记录。<router-link :to=’’replace>

router.go(n)参数是一个整数，意思是在 history 记录中向前或者后退多少步，类似 window.history.go(n)。

8.路由命名

例：routes: [

{

path: '/user/:userId',

name: 'user',

component: User

}

]

要链接到一个命名路由，可以给 router-link 的 to 属性传一个对象：

<router-link :to="{ name: 'user', params: { userId: 123 }}">User</router-link>

或：router.push({ name: 'user', params: { userId: 123 }})

这两种方式都会把路由导航到 /user/123 路径。

9.命名视图

有时候想同时（同级）展示多个视图，而不是嵌套展示你可以在界面中拥有多个单独命名的视图，而不是只有一个单独的出口。如果 router-view 没有设置名字，那么默认为 default。

10.重定向和别名

重定向也是通过 routes 配置来完成，下面例子是从 /a 重定向到 /b：

routes: [

{ path: '/a', redirect: '/b' }

]

重定向的目标也可以是一个命名的路由：

{ path: '/a', redirect: { name: 'foo' }}

别名

routes: [

{ path: '/a', component: A, alias: '/b' }

]

『别名』的功能让你可以自由地将 UI 结构映射到任意的 URL，而不是受限于配置的嵌套路由结构。

11.导航钩子（**『导航』表示路由正在发生改变**）

vue-router 提供的导航钩子主要用来拦截导航，让它完成跳转或取消。有多种方式可以在路由导航发生时执行钩子：全局的, 单个路由独享的, 或者组件级的

当一个导航触发时，全局的 before 钩子按照创建顺序调用。钩子是异步解析执行，此时导航在所有钩子 resolve 完之前一直处于 **等待中**。

const router = new VueRouter({…})

1.router.beforeEach(to,form,next)

to:Route: 即将要进入的目标 [**路由对象**](https://router.vuejs.org/zh-cn/api/route-object.html)

form:Route:  当前导航正要离开的路由

next:Function: 一定要调用该方法来 **resolve** 这个钩子。执行效果依赖 next 方法的调用参数:

* **next()**: 进行管道中的下一个钩子。如果全部钩子执行完了，则导航的状态就是 **confirmed** （确认的）。
* **next(false)**: 中断当前的导航。如果浏览器的 URL 改变了（可能是用户手动或者浏览器后退按钮），那么 URL 地址会重置到 from 路由对应的地址。
* **next('/') 或者 next({ path: '/' })**: 跳转到一个不同的地址。当前的导航被中断，然后进行一个新的导航。

**确保要调用 next 方法，否则钩子就不会被 resolved(释放)。**

2.router.afterEach()

after 钩子没有 next 方法，不能改变导航

3.在某个路由独享的钩子（在路由配置上直接定义 beforeEnter 钩子）

const router = new VueRouter({

routes: [

{

path: '/foo',

component: Foo,

beforeEnter: (to, from, next) => {

// ...

}

}

]

})

这些钩子与全局 before 钩子的方法参数是一样的。

4.组织内的钩子

beforeRouteEnch(to,form,next)

**不能** 访问 this，因为钩子在导航确认前被调用,因此即将登场的新组件还没被创建。

beforeRouteUpdate(to,form,next)

在当前路由改变，但是该组件被复用时调用 可以访问组件实例 `this`

beforeRouteLeave(to,form,next)

导航离开该组件的对应路由时调用 可以访问组件实例 `this`

12.路由元信息

meta: { requiresAuth: true }

我们称呼 routes 配置中的每个路由对象为 **路由记录**。路由记录可以是嵌套的，因此，当一个路由匹配成功后，他可能匹配多个路由记录

一个路由匹配到的所有路由记录会暴露为 $route 对象（还有在导航钩子中的 route 对象）的 $route.matched 数组。因此，我们需要遍历 $route.matched 来检查路由记录中的 meta 字段。

13.过渡动效

<transition>

<router-view></router-view>

</transition>

[**<transition> 的所有功能**](http://vuejs.org/guide/transitions.html) 在这里同样适用。

单个路由的过渡：在各路由组件内使用 <transition> 并设置不同的 name。

template: `

<transition name="slide">

<div class="foo">...</div>

</transition>

`

基于路由的动态过渡：基于当前路由与目标路由的变化关系，动态设置过渡效果（完整例子 [**这里**](https://github.com/vuejs/vue-router/blob/next/examples/transitions/app.js).）