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

bootcdn

<script src="https://cdn.bootcss.com/vue/2.5.17-beta.0/vue.js"></script>

<div id="app">

{{message}}

<h1>{{show("world!")}}</h1>

</div>

var app=new Vue({

el: "#app",

data: {

message: "Hello Vue"

},

methods: {

show: function(str){

return 'hello'+str+this.message;

}

}

})

控制台中app.message="123456"

则页面动态显示123456

现在数据和 DOM 已经被建立了关联，所有东西都是响应式的。

**el**: element 必须是html根容器元素

data存数据methods 存方法

属性绑定

<span v-bind:title="message">

v-bind 特性被称为指令。指令带有前缀 v-，以表示它们是 Vue 提供的特殊特性。它们会在渲染的 DOM 上应用特殊的响应式行为。在这里，该指令的意思是：“将这个元素节点的 title 特性值和 Vue 实例的 message 属性保持一致

title属性：鼠标移上去时显示的

<a v-bind:href="message">点击</a>

<input type="text" v-bind:value="message">

<div id="app-2">

<p v-html="m"></p>

</div>

var app2=new Vue({

el: "#app-2",

data: {

m: "<a href='https://www.baidu.com'> this is baidu</a>"

}

})

把m里的内容解析为一个html结构

**事件**

<button v-on:click="al">nihao</button>

<button @click="al">nihao</button>效果一样

methods: {

al: function(){

alert("nihaoa");

}

}

dblclick双击事件

事件修饰符

<button v-on:mousemove.stop="">阻止事件冒泡

v-on:click.once只触发一次的事件

v-on:click.prevent 阻止默认事件

提交事件不再重载页面  
<form v-on:submit.prevent="onSubmit"></form>

<form v-on:submit.prevent></form>

* 按键修饰符

只有在 `keyCode` 是 13 时调用vm.submit()

<input v-on:keyup.13="submit">

.enter

.tab

.delete (捕获“删除”和“退格”键)

.esc

.space

.up

.down

.left

.right

v-on:keyup.enter 按下enter键后触发的方法，就像qq输完密码按enter登录

* 系统修饰键

.ctrl

.alt

.shift

.meta

在 Mac 系统键盘上，meta 对应 command 键 (⌘)。在 Windows 系统键盘 meta 对应 Windows 键 (⊞)。

<!-- Alt + C -->

<input @keyup.alt.67="clear">

<!-- Ctrl + Click -->

<div @click.ctrl="doSomething">Do something</div>

请注意修饰键与常规按键不同，在和 keyup 事件一起用时，事件触发时修饰键必须处于按下状态。换句话说，只有在按住 ctrl 的情况下释放其它按键，才能触发 keyup.ctrl

**组件**

可复用的Vue实例

创造一个新Vue组件

Vue.component('button-counter', {

data: function () {

return {

count: 0

}

},

template: '<button v-on:click="countPlus">You clicked me {{ count }} times.</button>',

methods: {

countPlus: function(){

count++;

}

}

})

button-counter组件名，当自定义标签使用

没有el属性

data必须是一个函数。每个实例维护一份被返回对象的独立的拷贝

template有一个根标签，里面可以有很多标签。比如<p><button><div></div></p>

<div id="components-demo">

<button-counter></button-counter>

</div>

new Vue({ el: '#components-demo' })

computed计算属性

我们可以将同一函数定义为一个方法而不是一个计算属性。两种方式的最终结果确实是完全相同的。**然而，不同的是计算属性是基于它们的依赖进行缓存的。只在相关依赖发生改变时它们才会重新求值。**这就意味着只要 message 还没有发生改变，多次访问 reversedMessage 计算属性会立即返回之前的计算结果，而不必再次执行函数。

这也同样意味着下面的计算属性将不再更新，因为 Date.now() 不是响应式依赖：

computed: {

now: function () {

return Date.now()

}

}

相比之下，每当触发重新渲染时（就是Vue实例有数据更新），调用方法将总会再次执行函数。

我们为什么需要缓存？假设我们有一个性能开销比较大的计算属性 A，它需要遍历一个巨大的数组并做大量的计算。然后我们可能有其他的计算属性依赖于 A 。如果没有缓存，我们将不可避免的多次执行 A 的 getter！如果你不希望有缓存，请用方法来替代。

* v-once 指令，执行一次性地插值，当数据改变时，插值处的内容不会更新

<span v-once>这个将不会改变: {{ msg }}</span>

class与style绑定

<h1 v-bind:class="{red: true,blue: true}">helloworld</h1>

则在DOM中<h1 class="red blue">helloworld</h1>

class里面是一个对象，如果red属性值为true，则class中有属性red

* 动态添加样式

<h1 v-bind:class="{red: isRed}">helloworld</h1>

new Vue({

el: "#app",

data: {

isRed:false

},

methods: {

changeColor: function(){

this.isRed=!this.isRed;

}

}

})

我们也可以在这里绑定一个返回对象的计算属性。这是一个常用且强大的模式：

<div v-bind:class="classObject"></div>

data: {

isActive: true,

error: null

},

computed: {

classObject: function () {

return {

active: this.isActive && !this.error,

'text-danger': this.error && this.error.type === 'fatal'

}

}

}

v-if

<p v-if:"flag">flag=true时显示</p>

<p v-else-if:"flag2">flag=false, 且flag2=true时显示</p>

flag=false时第一个p就不存在，DOM中没有这个p

v-show

语法同上，隐藏时是display：none，结构还在DOM