# 一 组件的使用

组件就是一系列自定义的标签，组件的命名需要遵循驼峰命名（camelCase）、烤串命名（kebab-case）命名规范。推荐使用：my-component 这样的组件（标签）名。

注意：绑定属性一般也使用上述方式<my-comp :my-name=”test”></my-comp>，但是在props中接收该属性时，需要写为myName。

总结：无论是组件名还是属性名，html中使用 my-comp，js中使用myComp

## 1全局注册

全局注册的组件可以在任何模板中使用。使用方式如下：Vue.component(组件名,选项)

|  |
| --- |
| <**div id="app"**>  <**my-tab**></**my-tab**> </**div**>  <**script**>  **Vue**.**component**(**'my-tab'**,{ //必须在实例创建前注册！  //template的DOM结构必须被一个元素包含  **template**: **`<input type="button" value="自定义组件"/>`** })  **new Vue**({  **el**: **'#app'** }) </**script**> |

## 2 局部注册

局部注册的组件只有在该实例的作用域下有小，

|  |
| --- |
| <**div id="app"**>  <**my-tab**></**my-tab**> </**div**>  <**script**>  **new Vue**({  **el**: **'#app'**,  **components**: {  **'my-tab'**: {  **template**: **`<input type="button" value="自定义组件"/>`** }  }  }) |

## 3 is解除html限制

vue组件的模板会受到html本身语法的限制，比如<table>内只允许是<tr><td>等表格元素，在table内直接使用组件是错误的，这时候可以使用 is属性 来挂载组件。

<table><tbody is=”my-tab”></tbody></table>

此时tbody元素会被渲染为组件my-tab，类似的限制元素还有 ul ol select。

注意：如果是字符串模板时不受限制的。

## 4 组件的data

组件除了具备template属性外，也具备data、computed、methods等属性，但是和实例区别是，组件的data必须是函数，然后将数据返回出去。

额外注意：JS对象是引用关系，如果return出的对象引用了一个对象，那么这个对象就是共享的，任何一方修改都会造成同步，如下所示：

<div id="app">  
 <my-component></my-component>  
 <my-component></my-component>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 let data = {  
 counter: 0  
 };  
 Vue.component('my-component',{  
 template: '<button @click="counter++">{{counter}}</button>',  
 data: function () {  
 return data;  
 }  
 });  
 let app = new Vue({  
 el: '#app',  
 });  
</script>

点击任意一个按钮都会造成数字+1，因为组件的data引用来外部对象，我们需要给组件返回一个全新的data对象：

<div id="app">  
 <my-component></my-component>  
 <my-component></my-component>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 let data = {  
 counter: 0  
 };  
 Vue.component('my-component',{  
 template: '<button @click="counter++">{{counter}}</button>',  
 data: function () {  
 return {  
 counter: 0  
 };  
 }  
 });  
 let app = new Vue({  
 el: '#app',  
 });  
</script>

# 二 组件通信

## 1 父传子-props属性绑定

|  |
| --- |
| <**div id="app"**>  <**my-tab my-value="Test"**></**my-tab**> </**div**>  <**script**>  **new Vue**({  **el**: **'#app'**,  **components**: {  **'my-tab'**: {  **props**: [**'myValue'**],  **template**: **`<input type="text" :value="myValue"/>`** }  }  }) </**script**> |

当然大部分情况下传递的数据都是动态的，一般使用v-bind绑定：

<div id="app">  
 <input type="text" v-model="parentMsg">  
 <my-component :msg="parentMsg"></my-component>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 Vue.component('my-component',{  
 props: ['msg'],  
 template: '<span @click="counter++">{{msg}}</span>',  
 });  
 let app = new Vue({  
 el: '#app',  
 data: {  
 parentMsg:'默认'  
 }  
 });  
</script>

这里使用了v-model绑定父级数据parentMsg，输入任意的值，传递给子组件。

注意：如果不使用v-bind，仍然是采用类似第一个案例中  **my-value="Test"，**这时候，如果传递的是数字、数组、布尔值、对象，仅以字符串形式传递！

vue2与vue1的区别是：vue2通过props传递数据仅仅是单向的。在vue1中可以通过 .sync修饰符来支持双向绑定，这里不做介绍。

## 2 props数据验证

props的选项值可以是数组，也可以是对象，对象值通常用来对数据进行验证：

props: {

propA:Number, //必须是数字

propB:[String,Number], //必须是字符串或者数字

PropC:{ //必须是布尔，默认为false，且必须传入该数据

type:Boolean,

default:false，

required:true

},

propD:{

type:Array,

default:function(){ //默认值可以是个函数返回值

return [];

}

},

propF:{ //自定义雅正函数

validator:function(val){

retrun val > 10;

}

}

}

注意：type也可以是一个自定义的构造器，使用instanceof检测。如果props验证不通过，开发版本下控制台会抛出警告。

## 3 子传父

子组件向父组件传递数据需要用到自定义事件，子组件使用$emit()触发事件，父组件使用$on()监听子组件的事件。父组件也可直接在子组件上用v-on来监听子组件的自定义事件。

|  |
| --- |
| <**div id="app"**>  <**div**>  <**my-father :father-prop="list"**> </**my-father**>  </**div**> </**div**> <**script**>  **Vue**.**component**(**'my-father'**,{  data: **function** () {  **return** {  **val**: **''** }  },  **props**: [**'fatherProp'**],  **template**: **`  <div>  <h3>父组件</h3>  <input type="text" :value="val">  <my-son :son-prop="fatherProp" @receive="changeVal"></my-son>  </div>  `**,  **methods**: {  changeVal(value){  **this**.**val** = value  }  }  })   **Vue**.**component**(**'my-son'**,{  **props**:[**'sonProp'**],  **template**: **`  <div>  <h5>子组件</h5>  <ul>  <li v-for="item in sonProp" @click="chooseLi(item)">{{item}}</li>  </ul>  </div>  `**,  **methods**: {  chooseLi(item){  **this**.$emit(**'receive'**,item)  }  }  })   **new Vue**({  **el**: **'#app'**,  **data**: {  **list**: [**'a'**,**'b'**,**'c'**]  }  })  </**script**> |

## 4 非父子组件通信

### 4.1 方案一 vue1中的办法

vue1中使用$dispatch()向上派发事件，使用$broadcast()向下广播事件，这两种方法一旦发出事件后，任何组件都可以接收到，且遵循就近原则，在第一次接收到后停止冒泡，除非返回true。但是基于组件树结构的事件流方式在扩展性上很差，且不能解决兄弟组件间的通信问题，在vue2中被废弃。

### 4.2 方案二 中央事件总线bus

<div id="app">  
 内容为：{{msg}}  
 <my-component>组件</my-component>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 let bus = new Vue(); //随时引入一个Vue作为总线  
 Vue.component('my-component',{  
 template: '<button @click="handleEvent">点击传递事件</button>',  
 methods: {  
 handleEvent: function () {  
 bus.$emit('onMSG','from myComponent...');  
 }  
 }  
 });  
 let app = new Vue({  
 el: '#app',  
 data: {  
 msg: ''  
 },  
 mounted: function () {  
 let \_this = this;  
 bus.$on('onMSG',function (msg) {  
 \_this.msg = msg;  
 });  
 }  
 });  
</script>

创建一个类似中介的空vue实例bu，在生命周期函数中监听来自bus的onMSG事件，在回调函数中完成业务。如果深入使用，则可以给bus扩展data、methods、computed等，都可以公用，在业务中，如用户的登录昵称、性别、邮箱、授权token等都可以通过该方式实现。

### 4.3 方案三 状态管理vuex

大项目中使用该方式。

### 4.4 方案四 父链this.$parent

在子组件中，使用this.$parent可以直接访问该组件的父实例或组件，父组件也可以通过this.$children访问它所有的子组件，而且可以递归向上或向下无限访问，直到根实例或最内层的组件。但是实际开发中，这样做会让父子组件出现严重耦合，只看父组件，很难理解父组件的状态，因为它可能被任意组件修改，理想的情况应该是只有组件自己才能修改自己的状态。父子组件最好还是通过props和$emit来通信。

# 三 slot分发内容

## 1 单个slot

同时使用了父组件的内容、子组件的模板，被称为内容分发。

在子组件中使用特殊的slot标签作为内容的插槽。

单个slot：

如果父组件提供内容，则整个内容片段插入到slot所在的dom位置，并替换slot标签； 如果子组件模板没有slot标签，父组件提供的内容会被抛弃。

假定 子组件<son-component>组件有如下模板：

|  |
| --- |
| <div>  <slot>子组件：只有在没有要分发的内容时才会显示</slot> </div> |

父组件模板：

|  |
| --- |
| <div>  <son-component>  <p>父组件初始内容</p>  </son-component> </div> |

渲染结果：

|  |
| --- |
| <div>  <div>  <p>父组件初始内容</p>  </div> </div> |

编译作用域的注意点：

父组件有以下模板：<child-comp>{{msg}}</child-comp>

这里的msg就是一个slot，绑定的是父组件的数据，不是组件<child-comp>的数据。父组件模板的内容是在父组件作用域内编译，子组件模板的内容是在子组件作用域内编译。

## 2 具名slot

<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
 <style>  
 #div1 {  
 background-color: red;  
 }  
 </style>  
</head>  
<body>  
<div id="app">  
 <my-comp>  
 <h2 slot="header">标题</h2>  
 <p>正文内容</p>  
 </my-comp>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 Vue.component('my-comp',{  
 template: `  
 <div>  
 <div id="div1">  
 <slot name="header">slot</slot>  
 </div>  
 </div>`,  
 });  
 new Vue({  
 el: '#app'  
 });  
</script>

## 3 作用域插槽

作用域插槽是一个特殊的slot，使用一个可以复用的模板替换已渲染的元素。

案例一：

<div id="app">  
 <my-comp>  
 <template scope="props">  
 <p>来自父组件的内容</p>  
 <p>{{props.msg}}</p>  
 </template>  
 </my-comp>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 Vue.component('my-comp',{  
 template:  
 `<div class="container">  
 <slot msg="来自子组件的内容"></slot>  
 </div>`,  
 });  
 new Vue({  
 el: '#app'  
 });  
</script>

在子组件模板中，<slot>元素上有个类似props传递数据给组件的写法，msg=””，将数据传递给了插槽。父组件使用<template>元素，scop=’props’只是一个临时变量，类似v-for=”item in items”中的item一样，template内可以通过临时变量props访问来自子组件插槽的数据msg。

作用域插槽的代表案例：列表组件

<div id="app">  
 <my-comp :books="books">  
 <template slot="book" scope="props">  
 <li>{{props.bookName}}</li>  
 </template>  
 </my-comp>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 Vue.component('my-comp',{  
 props: {  
 books: {  
 type: Array,  
 default: function () {  
 return [];  
 }  
 }  
 },  
 template:  
 `<ul>  
 <slot name="book" v-for="book in books" :book-name="book.name"></slot>  
 </ul>`,  
 });  
 new Vue({  
 el: '#app',  
 data: {  
 books: [  
 {name: '《aaa》'},  
 {name: '《bbb》'},  
 {name: '《ccc》'}  
 ]  
 }  
 });  
</script>

子组件接收来自父级的props数组books，并且将它在name为book的slot上使用v-for循环，暴露一个边路昂bookName。其实这个案例中，直接在父级使用v-for就可以了，但是却在子组件中循环。针对该案例，确实多此一举，但是如果使用场景是既可以复用子组件的slot，又可以使slot内容不一致，上述案例还在其他组件内使用，<li>的内容就由使用者掌握的，数据可以通过临时变量props从子组件内获取。

## 4 访问slot

vue1中使用v-el间接访问，vue2中使用$slots访问。比如上面具名slot案例中的slot访问方式：this.$slots.header。

this.$slots.default包括了所有被包含在具名slot中的节点。

# 四 组件高级用法

## 1 递归组件

组件在它的模板内部可以递归的调用自己，只要给组件设置name的选项就可以。但是必须给一个条件来限制递归的数量，否则会抛出错误：max stackk size exceeded。

<div id="app">  
 <my-comp :count="1">  
 test  
 </my-comp>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 Vue.component('my-comp',{  
 name: 'my-comp',  
 props: {  
 count: {  
 type: Number,  
 default: 1  
 }  
 },  
 template:  
 `<div>  
 <my-comp :count="count + 1" v-if="count < 3"></my-comp>  
 </div>`,  
 });  
 new Vue({  
 el: '#app',  
 data: {  
 books: [  
 {name: '《aaa》'},  
 {name: '《bbb》'},  
 {name: '《ccc》'}  
 ]  
 }  
 });  
</script>

## 2 内联模板

如果给组件标签使用inline-template特性，组件就会把它的内容当做模板，而不是内容分发。父子组件的数据都会被渲染，由于这样做作用域非常不明显，不推荐使用。

## 3 动态组件

特殊标签<component>可以用来挂载不同的组件（利用is特性）。

<div id="app">  
 <component :is="current"></component>  
 <button @click="handleChangeView('A')">显示A</button>  
 <button @click="handleChangeView('B')">显示B</button>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 new Vue({  
 el: '#app',  
 components: {  
 comA: {  
 template: '<div>组件A</div>'  
 },  
 comB: {  
 template: '<div>组件B</div>'  
 }  
 },  
 data: {  
 current: 'comA'  
 },  
 methods: {  
 handleChangeView: function (component) {  
 this.current = 'com' + component;  
 }  
 }  
 });  
</script>

## 4 异步组件

项目太大时，一次性加载组件非常消耗性能。vue允许将组件定义为一个工厂函数，动态解析组件，当组件需要渲染时触发工厂函数，把结果缓存起来用于后面再次渲染。

<div id="app">  
 <my-comp></my-comp>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 Vue.component('my-comp',function (resolve,reject) {  
 window.setTimeout(function () {  
 resolve({  
 template:'<div>被异步渲染了</div>'  
 });  
 },2000)  
 });  
 new Vue({  
 el: '#app'  
 });  
</script>