

# 第七章 可复用性的组件详解



## 7.1 使用组件的原因

作用：提高代码的复用性

## 7.2 组件的使用方法

### 1. 全局注册

```
Vue.component('my-component',{  
  template:'<div>我是组件的内容</div>'  
})
```

优点：所有的vue实例都可以用

缺点：权限太大，容错率降低

### 2. 局部注册



```
var app = new Vue({  
  el: '#app',  
  components:{  
    'my-component':{  
      template: '<div>我是组件的内容</div>'  
    }  
  }  
})
```

3. vue组件的模板在某些情况下会受到html标签的限制，比如 `<table>` 中只能还有 `<tr>`，`<td>` 这些元素，所以直接在table中使用组件是无效的，此时可以使用is属性来挂载组件

```
<table>  
  <tbody is="my-component"></tbody>  
</table>
```

## 7.3 组件使用的奇淫技巧

1. 推荐使用小写字母加-进行命名（必须） child, my-componnet命名组件
2. template中的内容**必须**被一个DOM元素包括，也可以嵌套
3. 在组件的定义中，除了template之外的其他选项—data,computed,methods
4. data**必须**是一个方法

## 7.4 使用props传递数据 父亲向儿子传递数据

1. 在组件中使用props来从父亲组件接收参数，注意，在props中定义的属性，都可以在组件中直接使用
2. props来自父级，而组件中data return的数据就是组件自己的数据，两种情况作用域就是组件本身，可以在template, computed, methods中直接使用
3. props的值有两种，一种是字符串数组，一种是对象，本节先只讲数组
4. 可以使用v-bind动态绑定父组件来的内容

## 7.5 单向数据流

- 解释：通过 props 传递数据 是单向的了，也就是父组件数据变化时会传递给子组件，但是反过来不行。
- 目的：是尽可能将父子组件解耦，避免子组件无意中修改了父组件的状态。
- 应用场景：业务中会经常遇到两种需要改变 prop 的情况

一种是父组件传递初始值进来，子组件将它作为初始值保存起来，在自己的作用域下可以随意使用和修改。这种情况可以在组件 data 内再声明一个数据，引用父组件的 prop

步骤一：注册组件

步骤二：将父组件的数据传递进来，并在子组件中用props接收

步骤三：将传递进来的数据通过初始值保存起来

```
<div id="app">
  <my-comp init-count="666"></my-comp>
</div>
<script>
  var app = new Vue({
    el: '#app',
    components: {
      'my-comp': {
        props: ['init-count'],
        template: '<div>{{init-count}}</div>',
        data: function () {
          return {
            count: this.initCount
          }
        }
      }
    }
  })
```

```

    }
  }
}
})
</script>

```

另一种情况就是 prop 作为需要被转变的原始值传入。这种情况用计算属性就可以了

步骤一：注册组件

步骤二：将父组件的数据传递进来，并在子组件中用props接收

步骤三：将传递进来的数据通过计算属性进行重新计算

```

<input type="text" v-model="width">
<my-comp :width="width"></my-comp>
-----
var app = new Vue({
  el: '#app',
  data: {
    width: ''
  },
  components: {
    'my-comp': {
      props: ['init-count', 'width'],
      template: '<div :style="style">{{init-count}}</div>',
      computed: {
        style: function () {
          return {
            width: this.width + 'px',
            background: 'red'
          }
        }
      }
    }
  }
})

```

## 7.6 数据验证

@ vue组件中camelCased (驼峰式) 命名与 kebab-case (短横线命名)

@ 在html中, **myMessage** 和 **mymessage** 是一致的,,因此在组件中的html中使用必须使用kebab-case ( 短横线 ) 命名方式。在html中不允许使用驼峰!!!!!!

@ 在组件中, 父组件给子组件传递数据必须用短横线。在template中, 必须使用驼峰命名方式, 若为短横线的命名方式。则会直接报错。

@ 在组件的data中,用this.XXX引用时,只能是驼峰命名方式。若为短横线的命名方式, 则会报错。

验证的 type 类型可以是：

- String
- Number
- Boolean
- Object
- Array
- Function

```
Vue.component ( ' my-compoment ', {
  props : {
    // 必须是数字类型
    propA : Number ,
    // 必须是字符串或数字类型
    propB : [String , Number] ,
    // 布尔值, 如果没有定义, 默认值就是 true
    propC: {
      type : Boolean ,
      default : true
    },
    // 数字, 而且是必传
    propD: {
      type: Number ,
      required : true
    },
    // 如果是数组或对象, 默认值必须是一个函数来返回
    propE: {
      type : Array ,
      default : function () {
        return [] ;
      }
    },
    // 自定义一个验证函数
    propF: {
      validator : function (value) {
        return value > 10;
      }
    }
  }
})
```

```

    }
  }
}
});

```

## 7.7 组件通信

组件关系可分为父子组件通信、兄弟组件通信、跨级组件通信

### 7.7.1 自定义事件——子组件给父组件传递数据

使用v-on 除了监听 DOM 事件外，还可以用于组件之间的自定义事件。

JavaScript 的设计模式——观察者模式，dispatchEvent 和 addEventListener这两个方法。Vue 组件也有与之类似的一套模式，子组件用\$emit ( )来 触发事件，父组件用\$on ( )来 监听子组件的事件。

直接用代码

第一步：自定义事件

第二步：在子组件中用\$emit触发事件，第一个参数是事件名，后边的参数是要传递的数据

第三步：在自定义事件中用一个参数来接受

```

<div id="app">
  <p>您好,您现在的银行余额是{{total}}元</p>
  <btn-compnent
    @change="handleTotal"></btn-compnent>
</div>

<script src="js/vue.js"></script>
<script>
  //关于
  var app = new Vue({
    el:'#app',
    data:{
      total:0
    },
    components:{
      'btn-compnent':{
        template:'<div>\
          <button @click="handleincrease">+1</button> \
          <button @click="handlereduce">-1</button>\
        </div>',
        data:function(){
          return {
            count:0
          }
        }
      }
    }
  })

```

```

    },
    methods: {
      handleincrease :function () {
        this.count++;
        this.$emit('change',this.count);
      },
      handlereduce:function () {
        this.count--;
        this.$emit('change',this.count);
      }
    }
  },
  methods: {
    handleTotal:function (total) {
      this.total = total;
    }
  }
})

```

## 7.7.2 在组件中使用v-model

\$emit的代码,这行代码实际上会触发一个 input事件, 'input'后的参数就是传递给v-model绑定的属性的值

v-model 其实是一个语法糖, 这背后其实做了两个操作

- v-bind 绑定一个 value 属性
- v-on 指令给当前元素绑定 input 事件

要使用v-model,要做到:

- 接收一个 value 属性。
- 在有新的 value 时触发 input 事件

```

      _oo0oo_
      o8888888o
      88" . "88
      (| -_- |)
      0\  =  /0
      ___/'---'\___
      .' \\\|      |// '.
      / \\\||| : |||// \
      / _||| -:- ||||- \
      | _| \\\ - /// | _|
      | \_| ' '\---/' ' | _|
      \ .-\__ '- ' ___/-. /

```

```

      _ _ _ ' . . ' / - - . - \   ' . . ' _ _ _
      . " " ' <   ' . _ _ _ \ _ < | > _ / _ _ _ . ' > ' " " .
      | | :   ' - \ ` . ; \ _ / ` ; . \ / - ` : | |
      \ \ ` _ .   \ _ _ _ \ / _ _ /   . - ` /   /
===== ` - . _ _ _ _ ` . _ _ _ \ _ _ _ _ / _ _ _ . - ` _ _ _ . - ' =====
                        ` = _ _ _ _ = '

```

~~~~~

佛祖保佑                  永无BUG

```

<div id="app">
  <p>您好,您现在的银行余额是{{total}}元</p>
  <btn-compnent v-model="total"></btn-compnent>
</div>

<script src="js/vue.js"></script>
<script>
  //关于
  var app = new Vue({
    el:'#app',
    data:{
      total:0
    },
    components:{
      'btn-compnent':{
        template:'<div>\
          <button @click="handleincrease">+1</button> \
          <button @click="handlereduce">-1</button>\
        </div>',
        data:function(){
          return {
            count:0
          }
        },
        methods:{
          handleincrease :function () {
            this.count++;
            -----注意观察.这一行,emit的是input事件-----
            -
            this.$emit('input',this.count);
          },
          handlereduce:function () {
            this.count--;
            this.$emit('input',this.count);
          }
        }
      }
    }
  });

```

```

        }
    },
    methods: {
        /* handleTotal:function (total) {
            this.total = total;
        }*/
    }
})
</script>

```

### 7.7.3 非父组件之间的通信

官网描述：

有时候两个组件也需要通信(非父子关系)。在简单的场景下，可以使用一个空的 Vue 实例作为中央事件总线：

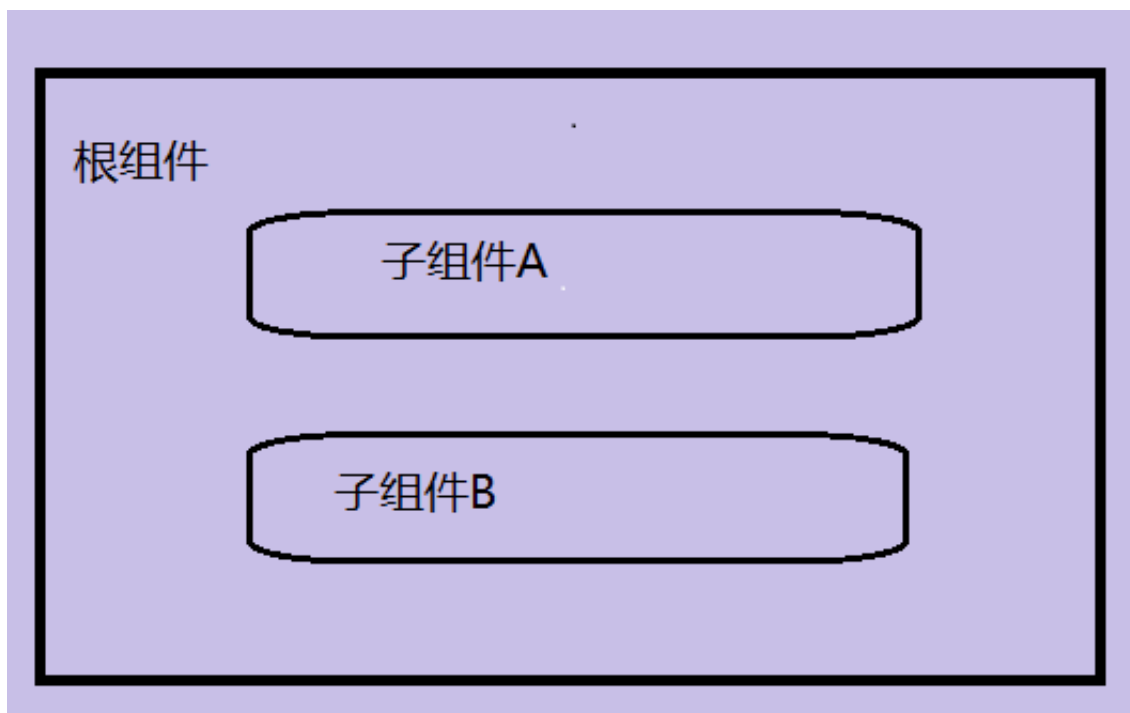
```
var bus = new Vue()
```

```
// 触发组件 A 中的事件
bus.$emit('id-selected', 1)
```

```
// 在组件 B 创建的钩子中监听事件
bus.$on('id-selected', function (id) {
    // ...
})
```

图形实例：





```
<div id="app" >
  <my-acomponent></my-acomponent>
  <my-bcomponent></my-bcomponent>

</div>

<script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js">
</script>
<script>

  Vue.component('my-acomponent',{
    template:'<div><button @click="handle">点击我向B组件传递数据</button></div>',
    data:function () {
      return{
        aaa:'我是来自A组件的内容'
      }
    },
    methods:{
      handle:function () {
        this.$root.bus.$emit('lala',this.aaa);
      }
    }
  })
  Vue.component('my-bcomponent',{
    template:'<div></div>',
    created:function () {
      //A组件在实例创建的时候就监听事件---lala事件
      this.$root.bus.$on('lala',function (value) {
        alert(value)
      })
    }
  })
}
```

```

    });
  }
})

```

父链：this.\$parent

```

Vue.component('child-component',{
  template:'<button @click="setFatherData">通过点击我修改父亲的数据</button>',
  methods:{
    setFatherData:function () {
      this.$parent.msg = '数据已经修改了'
    }
  }
})

```

子链：this.\$refs

提供了为子组件提供索引的方法，用特殊的属性ref为其增加一个索引

```

var app = new Vue({
  el:'#app',
  data:{
    //bus中介
    bus:new Vue(),
    msg:'数据还未修改',
    formchild:'还未拿到'
  },
  methods:{
    getChildData:function () {
      //用来拿子组件中的内容 ---- $refs
      this.formchild = this.$refs.c.msg;
    }
  }
})

```

## 7.8使用slot分发内容

### 7.8.1 什么是slot(插槽)

为了让组件可以组合，我们需要一种方式来混合父组件的内容与子组件自己的模板。这个过程被称为 **内容分发**。Vue.js 实现了一个内容分发 API，使用特殊的 'slot' 元素作为原始内容的插槽。

## 7.8.2 编译的作用域

在深入内容分发 API 之前，我们先明确内容在哪个作用域里编译。假定模板为：

```
<child-component>
  {{ message }}
</child-component>
```

message 应该绑定到父组件的数据，还是绑定到子组件的数据？答案是父组件。组件作用域简单地说是：

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。

## 7.8.3 插槽的用法

父组件的内容与子组件相混合，从而弥补了视图的不足

**混合父组件的内容与子组件自己的模板**

单个插槽：

```
<div id="app" >
  <my-component>
    <p>我是父组件的内容</p>
  </my-component>
</div>

Vue.component('my-component',{
  template:'<div>\
    <slot>\
      如果父组件没有插入内容，我就作为默认出现\
    </slot>\
  </div>'
})
```

具名插槽：

具名插槽：

```
<name-component>
  <h3 slot="header">我是标题</h3>
  <p>我是正文内容</p>
  <p>正文内容有两段</p>
  <p slot="footer">我是底部信息</p>
</name-component>
```

```
Vue.component('name-component',{
  template:'<div>\
```

```

        <div class="header">\n' +
    '    <slot name="header">\n' +
    '        \n' +
    '    </slot>\n' +
    '</div>\n' +
    '<div class="contatiner">\n' +
    '    <slot>\n' +
    '        \n' +
    '    </slot>\n' +
    '</div>\n' +
    '<div class="footer">\n' +
    '    <slot name="footer">\n' +
    '\n' +
    '    </slot> \n' +
    '</div>' +
    '    </div>'
  })

```

## 7.8.4 作用域插槽

作用域插槽是一种特殊的slot，使用一个可以复用的模板来替换已经渲染的元素——从子组件获取数据

====template模板是不会被渲染的



```

Vue.component('my-component',{
  template:'<div>\n
    <slot text="我是来自子组件的数据" ss="fdjkfjlsd" name="abc"
  >\n
    </slot>\n
  </div>'
})

```

## 7.8.5 访问slot

通过this.\$slots.(NAME)

```

mounted:function () {
  //访问插槽
  var header = this.$slots.header;
  var text = header[0].elm.innerText;
  var html = header[0].elm.innerHTML;
  console.log(header)
  console.log(text)
  console.log(html)
}

```

```
}
```

## 7.9 组件高级用法-动态组件

VUE给我们提供 了一个元素叫component

作用是： 用来动态的挂载不同的组件

实现：使用is特性来进行实现的