项目六 Vue组件

本章内容

- 6.1 组件的基本使用
- 6.2 Vue组件嵌套
- 6.3 组件通信
- 6.4 案例实战——**简单的照片相册**

6.1 组件的基本使用

- 6.1.1 什么是组件
- 6.1.2 组件的注册
- 6.1.3 组件中的 data 必须是函数

6.1.1 什么是组件

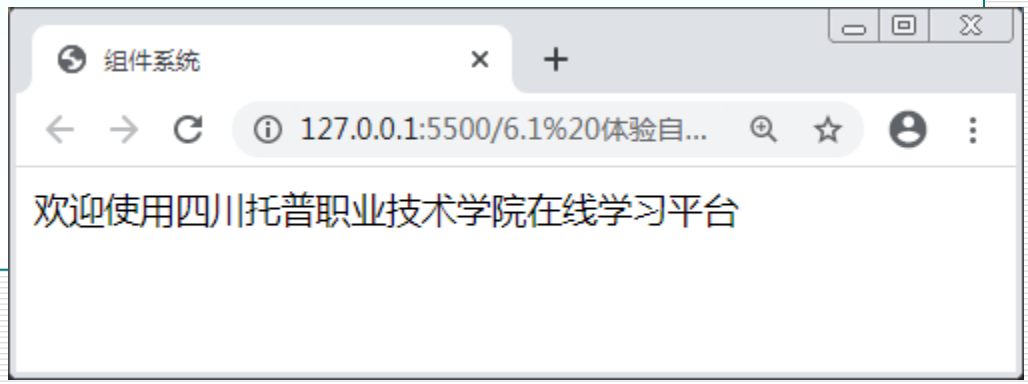
- 组件是Vue中的一个重要概念，它是一种抽象，是一个可以复用的Vue实例，它拥有独一无二的组件名称，它可以扩展HTML元素，以组件名称的方式作为自定义的HTML标签。因为组件是可复用的Vue实例，所以它们与new Vue()接收相同的选项，例如data、computed、watch、methods以及生命周期钩子等。仅有的例外是像el这样根实例特有的选项。

6.1.1 什么是组件

- 例如，在一个绝大多数的系统网页中，网页都包含header、menu、body、footer等部分，在很多时候，同一个系统中的多个页面，可能仅仅是页面中body部分显示的内容不同，因此，我们就可以**将系统中重复出现的页面元素设计成一个个的组件，当需要使用到的时候，引用这个组件即可。**
- 不过，与在编写C#时对代码进行模块化的划分 不同，模块化主要是为了实现每个模块、方法的职能单一，一般是通过代码逻辑的角度进行划分；而**Vue中的组件化**，更多的**是为了实现对于前端UI组件的重用。**

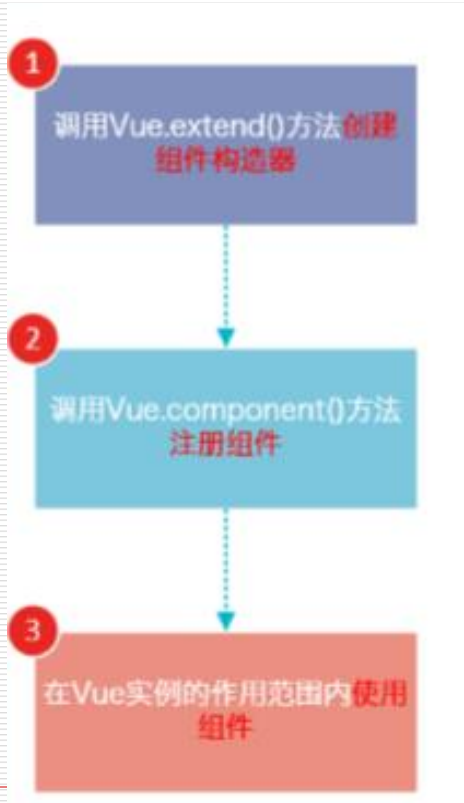
例6.1 体验自定义组件

```
<body>
    <div id="app">
        <!-- 3. #app是Vue实例挂载的元素，应该在挂载元素范围内使用组件-->
        <my-component></my-component>
    </div>
</body>
<script>
    // 1.创建一个组件构造器
    var myComponent = Vue.extend({
        template: '<div>欢迎使用四川托普职业技术学院在线学习平台</div>'
    })
    // 2.注册组件，并指定组件的标签，组件的HTML标签为<my-component>
    Vue.component('my-component', myComponent)
    new Vue({
        el: '#app'
    });
</script>
```



6.1.2 组件注册

Vue.js 的组件使用有三个步骤，具体步骤如下：



1. 调用 Vue.extend() 方法创建组件构造器

```
var MyComponent = Vue.extend({
   // 选项
})
```

2. 调用 Vue.component() 方法注册组件。

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

3. 在 Vue 实例的作用范围使用组件。

```
<div id="app">
   <my-component></my-component>
</div>
```

例6.2 演示自定义组件

```
<div id="app">
    <ol>
      <!-- 创建一个 todo-item 组件的实例 -->
      <todo-item ref="t1"></todo-item>
      <todo-item ref="t2"></todo-item>
      <todo-item ref="t3"></todo-item>
    </ol>
  </div>
  <script> // 定义名为 todo-item 的新组件. 组件实质上是对一个HTML片段的抽象。
    Vue.component('todo-item', {
      template: `
      <div>
        <li>这是个待办项</li>
        <p>代办人：{{name}}，代办详情：{{memo}}</p>
      </div>`,
      data: function () {
        var data = {
          name: "李四",
          memo: "办理大学毕业证"
        }
        return data;
      }
    })
    /*  new Vue() 此代码必须有，否则DOM元素无法通过Vue进行渲染操作和捆绑    */
    var app = new Vue({
      el: '#app'
    })
  </script>
```

6.1.2 组件注册

- 在Vue中创建一个新的组件之后，为了能在模板中使用，这些组件必须先进行注册，以便Vue能够识别。在Vue中有两种组件的注册类型：**全局组件和局部组件**。
- 全局注册的组件，可以用在通过new Vue( )新创建的Vue根实例中，也可以在组件树中的所有子组件的模板中使用；
- 局部注册的组件只能在当前注册的Vue实例中进行使用。

6.1.2 组件注册

- 全局组件
  - 在Vue中创建全局组件，通常的做法是先使用Vue.extend方法构建模板对象，然后通过Vue.component方法来注册组件。因为组件最后会被解析成自定义的HTML代码，因此，可以直接在HTML中使用组件名称作为标签来使用。

例6.3 全局注册组件

```
<div id="app">
    <my-component></my-component>
</div>
<script>
    //1、使用 Vue.extend 构建模板对象
    var comElement = Vue.extend({
        template: '<div><h3>全局组件</h3><p>这是我们创建的全局组件
</p></div>'
    })
    //2、使用 Vue.component 注册全局组件
    Vue.component('myComponent', comElement)
    var app = new Vue({
        el: '#app'
    });
</script>
```

例6.4 组件命名示例

```
<div id="app">
    <!--小驼峰命名的组件，使用方式-->
    <my-com></my-com>
</div>
<script>
    var comElement = Vue.extend({
        template: '<h3>组件名称使用的规则</h3>'
    })
    //小驼峰命名组件
    Vue.component('myCom', comElement)
    var app = new Vue({
        el: '#app'
    });
</script>
```

注：当采用小驼峰（myCom）的方式命名组件时，在使用这个组件的时候，需要将大写字母改成小写字母，同时两个字母之间需要使用“-”进行连接，例如<my-com>

例6.5 以匿名对象的方式注册组件

```
<div id="app">
    <my-com2></my-com2><br/>
    <my-com3></my-com3>
</div>
<script>
    Vue.component('myCom2', Vue.extend({
        template: '<div>这是直接使用 Vue.component 创建的组件myCom2</div>'
    }))
    Vue.component('myCom3', {
        template: '<div>这是直接使用 Vue.component 创建的组件myCom3</div>'
    })
    var app = new Vue({
        el: '#app'
    });
</script>
```

例6.6 错误的编写方式

```
<div id="app">
    <my-com></my-com>
</div>
<script>
    var comElement = Vue.extend({
        template: '<h3>全局组件</h3><p>这是我们创建的全局组件</p>'
    })
    Vue.component('my-com', comElement)
    var app = new Vue({
        el: '#app'
    });
</script>
```

6.1.2 组件注册

- template属性指向的模板内容可能包含多个元素，而使用Vue.extend创建的模板必须有且只有一个根元素，出现多个根元素时，默认只渲染第一个根元素的内容。因此，当需要创建具有复杂元素的模板时，可以在最外层再套一个div。

例6.6 应改为：

```
template: '<div><h3>全局组件</h3><p>这是我们创建的全局组件</p></div>'
```

6.1.2 组件注册

- 当template属性中包含很多的元素时，不能使用代码提示还是会显得不方便，可以使用template标签来定义模板，通过id来确定组件的模板信息。
- 例6.8 定义模板

```
<div id="app">
    <my-com></my-com>
</div>
<template id="tmp">
    <div>
        <h3>Vue.js</h3>
        <h4>是现今最流行的框架之一</h4>
    </div>
</template>
<script>
    Vue.component('my-com',{
        template: '#tmp'
    })
    var vm = new Vue({
        el: '#app',
    });
</script>
```

6.1.2 组件注册

- 局部组件
  - 有些时候，注册的组件只想在一个Vue实例中使用，如果还是使用全局注册的方式注册组件，就不太合适了。这时，可以使用局部注册的方式注册组件。
  - 在Vue实例中，可以通过components属性注册仅在当前作用域下可用的组件。

例6.9 局部注册组件

```
<div id="app1">
    <my-com></my-com>
</div>
<div id="app2">
    <my-com></my-com>
</div>
<template id="app2-com">
    <h4>app2中注册的局部组件</h4>
</template>
<script>
    var app1 = new Vue({
        el: '#app1',
    });
    var app2 = new Vue({
        el: '#app2',
        components: {
            'my-com': {
                template: '#app2-com'
            }
        }
    });
</script>
```

6.1.3 组件中的 data 必须是函数

- Vue 组件中 data 选项为什么必须是函数?
  - 因为一个组件可以在多处复用，如果 data 是一个对象，那么所有复用的组件实例将都显示相同内容，如此就限制了组件复用的意义。
  - 构造 Vue 实例时传入的各种选项大多数都可以在组件里使用。只有一个例外：data 必须是函数。

例6.10 演示组件中的data必须是函数

```
<h1>组件声明中，data必须是 <b>函数</b></h1>
    <div id="app">
        <h2>错误用法：data不是函数时</h2>
        <error-counter></error-counter>
        <error-counter></error-counter>
        <error-counter></error-counter>
        <h2>正确用法：data是函数时</h2>
        <right-counter></right-counter>
        <right-counter></right-counter>
        <right-counter></right-counter>
    </div>
```

```
<script>
    var data = {
        counter: 0
    }
    Vue.component('error-counter', {
        template: '<button v-on:click="counter += 1">{{ counter }}</button>',
        // 技术上 data 的确是一个函数了，因此 Vue 不会警告，
        // 但是我们返回给每个组件的实例的却引用了同一个data对象
        data: function () {
            return data;
        }
    })
    Vue.component('right-counter', {
        template: '<button v-on:click="counter += 1">{{ counter }}</button>',
        // 技术上 data 的确是一个函数了，因此 Vue 不会警告，
        // 但是我们返回给每个组件的实例的却引用了同一个data对象
        data: function () {
            return {
                counter: 0
            }
        }
    })
    //  初始化根实例
    var app = new Vue({
        el: '#app'
    })
</script>
```

6.2 Vue 组件嵌套

- 6.2.1 组件嵌套
- 6.2.2 使用props选项
- 6.2.3 插槽
- 6.2.4 组件实战

6.2.1 组件嵌套

- Vue 两大核心思想
  - 组件化：将一个整体合理拆分为各个小块(组件)，组件可复用。
  - 数据驱动：释放了对DOM的操作，让DOM随着数据的变化自然而然地变化，不过多 关注DOM，只需要将数据组织好即可。

6.2.1 组件嵌套

- 组件本身也可以包含组件

需要注意的是从Vue2.0开始，每个组件必须只有一个根元素。不再允许片段实例。

```
var Child = Vue.extend({
        template: '<div>我是子组件!</div>'
});
var Parent = Vue.extend({
    template: '<div>我是父组件 <child-component></child-component></div>',
   components: {
        'child-component': Child
   }
});
Vue.component("parent-component", Parent);
```

例6.11 演示嵌套组件

```
<body>
    <div id="app">
        <parent-component>
        </parent-component>
    </div>
</body>
<script>
    //子组件构造器
    var Child = Vue.extend({
        template: '<p>我是子组件!</p>'
    })
    //父组件构造器
    var Parent = Vue.extend({
        // 在Parent组件内使用<child-component>标签
        template: '<div>我是父组件<child-component></child-component> </div>',

        //引用子组件
        components: {
            // 局部注册Child组件，该组件只能在Parent组件内使用
            'child-component': Child
        }
    })
    // 全局注册Parent父组件
    Vue.component('parent-component', Parent)
    new Vue({
        el: '#app'
    })
</script>
```

6.2.2 使用props选项

- 组件实例的作用域是孤立的。这意味着不能并且不应该在子组件的模板内直接引用父组件的数据。通常使用 props 把数据传给子组件。
- 例6.12 使用 props 把数据传给子组件

```
<h1>使用 Prop 向组件传递数据</h1>
<div id="app">
    <child message="hello!" ref="child1"></child>
</div>
<script>
    Vue.component('child', {
        // 声明 props
        // 就像 data 一样，prop 可以用在模板内
        // 同样也可以在 vm 实例中像 “this.message” 这样使用
        props: ['message'],
        template: '<span>{{ message }}</span>'
    })
    //  初始化根实例
    var app7 = new Vue({
        el: '#app'
    })
</script>
```

6.2.2 使用props选项

- 在父组件引用子组件的时候，通过属性绑定的方式（v-bind），将需要传递给子组件的数据进行传递，从而在子组件内部，通过绑定的属性值获取到父组件的数据。

6.2.2 使用props选项

- 在父组件引用子组件的时候，通过属性绑定的方式（v-bind），将需要传递给子组件的数据进行传递，从而在子组件内部，通过绑定的属性值获取到父组件的数据。

例6.13 测试父组件引用子组件

```
<div id="app">
    <h4>
        请输入需要传递给子组件的值：<input type="text" v-model="title" />
    </h4>
    <child-node v-bind:parenttitle="title"></child-node>
</div>
<template id="child">
    <div>
        <h4>Vue 实例中的属性值为：{{content}}</h4>
    </div>
</template>
<script>
    var vm = new Vue({
        el: '#app',
        data: {
            title: ''
        },
        components: {
            'childNode': {
                template: '#child',
                props: ['parenttitle'],
                data() {
                    return {
                        content: this.parenttitle
                    }
                }
            }
        },
    });
</script>
```

例6.14 watch监听

```
<div id="app">
    <h4>
        请输入需要传递给子组件的值：<input type="text" v-model="title" />
    </h4>
    <child-node v-bind:parenttitle="title"></child-node>
</div>
<template id="child">
    <div>
        <h4>Vue 实例中的属性值为：{{content}}</h4>
    </div>
</template>
<script>
    var vm = new Vue({
        el: '#app',
        data: {
            title: ''
        },
        components: {
            'childNode': {
                template: '#child',
                props: ['parenttitle'],
                data() {
                    return {
                        content: this.parenttitle
                    }
                },
                watch: {
                    parenttitle: function () {
                        this.content = this.parenttitle
                    }
                },
            }
        },
    });
</script>
```

例6.15 动态 props 向子组件动态传递数据

```
<div id="app">
    输入信息：<input type="text" v-model="msg">
    <child v-bind:message="msg" v-bind:name="person" ref="child1"></child>
</div>
<script>
    Vue.component('child', {
        // 声明 props
        // 就像 data 一样，prop 可以用在模板内
        // 同样也可以在 vm 实例中像 “this.message” 这样使用
        props: ['message', 'name'],
        template: '<span>{{ message }} 发送者：{{name}}</span>'
    })
    //  初始化根实例
    var app = new Vue({
        el: '#app',
        data: {
            msg: 'hello',
            person: '四川托普'
        }
    })
</script>
```

6.2.3 插槽

- 插槽（slot），它是组件的一块HTML模板，这块模板显示不显示以及怎样显示由父组件来决定。
- 本小节主要讲解：
  - 单 Slot 插槽
  - 具名Slot 插槽
  - Slot 作用域插槽。

6.2.3 插槽

- 单 slot 插槽
  - 默认父组件在子组件内套的内容是不显示的。除非子组件模板包含至少一个<slot>插口，否则父组件的内容将会被丢弃。

例6.16 单 slot 插槽

```
<div id="app">
        <children>
            <!--span这行不会显示-->
            <span>注册成功</span>
        </children>
    </div>
    <script>
        var vm = new Vue({
            el: '#app',
            components: {
                children: {
                    // template: "<button>这是子组件</button>"
                    template: "<div><slot><p>默认效果</p></slot>这里是子组件
</div>"
                }
            }
        });
    </script>
```

6.2.3 插槽

- 具名slot 插槽
  - 如果一个组件中想使用多个slot，就需要使用具名slot。<slot>元素可以用一个特殊的属性name来配置分发内容。多个slot可以有不同的名字。
  - 具名slot将匹配模板内容片段中有对应slot特性的元素。

例6.17 具名 slot 插槽

```
<h1>具名slot插槽</h1>
    <p>slot 元素可以用一个特殊的属性 name 来配置如何接收父组件的分发内容。多个 slot 可以有不同的名字。具名 slot 将匹配内容片段中有对应 slot 特性的元素。 仍然可以有一个匿名 slot， 它是默认 slot
        ，作为找不到匹配的内容片段的备用插槽。如果没有默认的 slot， 这些找不到匹配的内容片段将被抛弃。 </p>

    <div id="app-7">
        <my-component>
            <h1 slot="header">页面标题</h1>
            <p>主要内容的一个段落。 </p>
            <p>另一个主要段落。 </p>
            <div slot="footer">
                <address>这里有一些联系信息</address>
            </div>
        </my-component>
    </div>
```

```
<script>
        Vue.component('my-component', {
            template: `
        <div class="container">
          <header>
            <slot name="header"></slot>
          </header>
          <main>
            <slot></slot>
          </main>
          <footer>
            <slot name="footer"></slot>
          </footer>
        </div>`
        })
        //  初始化根实例
        var app7 = new Vue({
            el: '#app-7'
        })
    </script>
```

6.2.3 插槽

- slot作用域插槽
  - 在父级中，具有特殊属性scope的template元素，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prop对象。

例6.18 slot作用域插槽

```
<h1>slot作用域插槽</h1>
    <p>在父级中，具有特殊属性 scope 的 template 元素，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 prop 对象。</p>

    <div id="app">
        <my-component>
            <template scope="myProps">
                <span>这里是父组件传入的信息！ </span>
                <span>这里是父组件从子组件接收到的数据，{{ myProps.text }}，格式化后再分发给插槽。</span>
            </template>
        </my-component>
    </div>
    <script>
        Vue.component('my-component', {
            template: `
        <div class="container">
            <slot text="hello from child"></slot>
        </div>`
        })

        //  初始化根实例
        var app7 = new Vue({
            el: '#app'
        })
    </script>
```

例6.19 slot作用域插槽列表组件

```
<h1>slot作用域插槽</h1>
    <p>在父级中，具有scope特殊属性的 template 元素，可以用于接收子组件传递出来的数据。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 prop
        对象。具有scope属性的template称为“作用域插槽模板”。</p>

    <div id="app">
        <my-component :items="myItems">
            <!--作用域插槽也可以是具名的 -->
            <template slot="item" scope="props">
                <!-- 允许父组件向子组件分发内容 -->
                <li class="my-fancy-item">{{props.username}} {{ props.text }}</li>
            </template>
        </my-component>
    </div>
```

```
<script>
        Vue.component('my-component', {
            props: ["items"],
            template: `
                <ul>
                    <hr>
                    <slot name="item" v-for="item in items" :username="item.username"
:text="item.text"></slot>
                    <hr>
                </ul>`,
            created: function () {
                console.log(this.items);
            }
        })

        //  初始化根实例
        var app = new Vue({
            el: '#app',
            data: {
                myItems: [
                    { username: '小慧', text: '毕业了' },
                    { username: '小兰', text: '毕业了' },
                    { username: '小强', text: '毕业了' }
                ]
            }
        })
    </script>
```

6.2.4 组件实战

- 例6.20 组件实战

name	age	sex
小明	20	男
小强	21	男
小兰	22	女
小惠	20	女

6.3 组件通信

- 6.3.1 父组件向子组件通信
- 6.3.2 子组件向父组件通信
- 6.3.3 任意组件及平行组件通信

6.3 组件通信

- 组件间的通信主要有 4 种 Vue 组件通信方式
  - **父子组件的通信**
  - **非父子组件的 eventBus 通信**
  - 利用本地缓存实现组件通信
  - Vuex 通信

6.3.1 父组件向子组件通信

- 父组件向子组件传递数据 的方式
  - 使用Props（默认是单向绑定）

1、使用 props 属性父组件向子组件传值可以使用如下代码：

<child-component v-bind:子组件属性="父组件数据属性"></child-component>

  - $parent

2、通过$parent 直接在子组件中通过this.$parent 的到调用其父组件，但并不建议使用

例6.21 父组件向子组件传递数据

```
<div id="app">
    <h4>
        请输入需要传递给子组件的值：<input type="text" v-model="title" />
    </h4>
    <child-node v-bind:parenttitle="title"></child-node>
</div>
<template id="child">
    <div>
        <h5>
            Vue实例中的属性值为：{{content}}
        </h5>
    </div>
</template>
```

```
<script>
    var app= new Vue({
        el: '#app',
        data: {
            title: ''
        },
        components: {
            'childNode': {
                template: '#child',
                props: ['parenttitle'],
                data() {
                    return {
                        content: this.parenttitle
                    }
                },
                watch: {
                    parenttitle() {
                        this.content = this.parenttitle
                    }
                },
            }
        }
    })
</script>
```

6.3.2 子组件向父组件通信

- 子组件向父组件通信的方式
  - 自定义事件
  - 使用$refs

6.3.2 子组件向父组件通信

- 使用自定义事件
  - 在父组件中调用子组建的时候，绑定一个自定义事件和对应的处理函数。

```
在 templete 里应用子组件时,定义事件 changeMsg
    <counter   @changeMsgEvent="changeMsg"></counter> // 2. 用 changeMsg 监听事件是否触发
    methods: {
      changeMsg:function(msg){    //msg 就是传递来的数据
      }
    }
```

  - 在子组件中把要发送的数据通过触发自定义事件传递给父组件。

```
this.$emit(“changeMsg ”,“这是子组件传过来的值”);
```

例6.22 子组件向父组件通信

```
<div id="container">
        <p>{{msg}}</p>
        <parent-component></parent-component>
    </div>
    <script>
        //通过事件的方式传递
        //    绑定 -- 触发
        Vue.component("parent-component", {
            data: function () {
                return {
                    sonMsg: ""
                }
            },
            methods: {
                //msg参数要拿子传递的值
                recvMsg: function (msg) {
                    console.log("父组件接收到子组件的数据" + msg);
                    this.sonMsg = msg;

                }
            },
```

```
template: `
        <div>
            <h1>这是父组件</h1>
            <p>子组件传来的数据为：{{sonMsg}}</p>
            <hr/>
            <child-component @customEvent="recvMsg"></child-component>
        </div>
    `
})
Vue.component("child-component", {
    methods: {
        sendMsg: function () {
            //来触发绑定给子组件的自定义方法
            //第一个参数触发,第二个参数传值
            this.$emit("customEvent", "Vue组件学习中");
        },
    },
    template: `
        <div>
            <h1>这是子组件</h1>
            <button @click="sendMsg">senToFather</button>
        </div>
    `
})
new Vue({
    el: "#container",
    data: {
        msg: "Hello VueJs"
    }
})
</script>
```

6.3.2 子组件向父组件通信

- 使用$refs
  - 在调用子组件的时候，可以制定refs 属性。

```
<child-component refs="xiaoming"></child-component>
```

  - 通过$refs 得到指定引用名称对应的组件实例

```
This.$refs.xiaoming
```

6.3.3 任意组件及平行组件通信

- eventBus这种通信方式针对的是非父子组件之间的通信，它的原理还是通过事件的触发和监听。但是因为非父子子组件的关系，它们需要有一个中间组件来连接。

6.3.3 任意组件及平行组件通信

使用eventBus 传递数据具体三个步骤。

```
// 1. 创建一个 Vue 实例
        bus : new Vue()
// 2.在子组件触发自定义的事件 $emit()——把事件沿着作用域链向上派送
       bus.$emit('changeMsgEvent', '需要传递的数据')
 // 3.在接收组件监听事件,接受数据
     mounted(){
      bus.$on('changeMsgEvent', function(msg){  //msg  是通过事件传递来的数据
 })
}
```

例6.23 任意组件及平行组件通信

```
<h1>组件之间的通信</h1>
   <p>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</p>

   <div id="app">
      <h2>组件A：向总线上报事件</h2>
      <my-component-a v-bind:counter="total"></my-component-a>
      <h2>组件B：通过总线监听相关事件</h2>
      <my-component-b></my-component-b>
   </div>
   <script>
      var bus = new Vue();

      Vue.component('my-component-a', {
         template: '<div><p>组件A</p><hr> <button v-on:click="doClick">{{ counter }}</button><hr></div>',
         data: function () {
            return { counter: 1 }
         },
         methods: {
            doClick: function () {
               this.counter++;
               bus.$emit('btn-click', this.counter)
            }
         }
      })
```

```
    Vue.component('my-component-b', {
        template: '<div><p>组件B</p><hr> 计数器：{{ counter }} <hr></div>',
        data: function () {
            return {
                counter: 0
            }
        },
        methods: {
            foo: function (value) {
                console.log(value);
                this.counter = value;
            }
        },
        created: function () {
            bus.$on('btn-click', this.foo);
        }
    })
    //  初始化根实例
    var app7 = new Vue({
        el: '#app',
        data: {
            total: 0
        },
        methods: {
            doChildClick: function () {
                this.total += 1
            }
        }
    })
</script>
```

6.4 案例实战——简单的照片相册

- 本案例是一个简单的相册展示页面，类似于轮播图效果。主要实现的功能如下：
- （1）可使用左右图标箭头切换图片。
- （2）可使用键盘上的左右方向键切换图片。
- （3）可通过下面的缩略图选择指定图片。

本章小结

- 本章主要介绍 Vue 组件知识，包括组件的定义，组件的作用域
- 组件中 data 必须是函数，组件嵌套，使用 props 向子组件传值
- props 验证数据的类型是否合法。组件之间的通信方式，$emit()——把事件沿着作用域链向上派送，$on()——监听事件
- 通过自定义组件并把组件打包发布深入学习了组件，为以后工程化项目开发积累了更多的经验。