

第5章



Vue.js 组件开发

本章学习目标

通过本章的学习,能够熟悉组件的命名规范,掌握组件注册方法;掌握组件间常用的通信方法;掌握插槽的分类和定义方法;能够在实际工程中使用插槽传递数据。

Web 前端开发工程师应知应会以下内容:

- 熟悉组件的命名规范;
- 掌握全局、局部注册组件的方法;
- 掌握组件间常用的通信方法;
- 掌握插槽的分类和定义方法。

5.1 组件基础

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。组件也可以看作自定义的 HTML 元素。可以使用独立可复用的小组件构建大型应用,几乎任意类型应用的界面都可以抽象为一棵组件树,如图 5-1 所示。

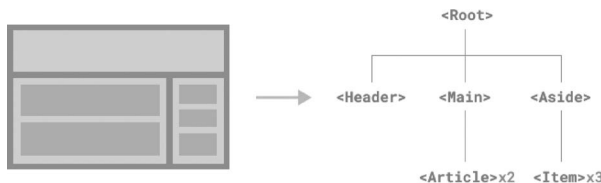


图 5-1 组件树

5.1.1 组件命名

注册组件前,必须给组件命名,组件名应该采用多个单词构成,以免与现有的或未来的 HTML 元素有冲突,由于所有 HTML 元素名称都是由单个单词构成。Vue 组件中通常采用 kebab-case(短横线分隔式)或 camelCase(驼峰式)命名方式。

1. kebab-case 命名方式

当定义一个组件时,必须在引用自定义元素时使用 kebab-case 命名方式。Vue props 最好直接使用 kebab-case 方式命名,如 my-name、my-component-name 等。

2. camelCase 命名方式

camelCase(也称为驼峰式)命名方式就是当组件名称是由一个或多个单词构成时,第 1 个单词首字母小写,从第 2 个单词开始的每个单词的首字母都采用大写,如 myFirstName、myLastName、myComponent。注意,尽管如此,直接在 DOM(即非字符串的模板)中使用时,

只有 kebab-case 形式的名称是有效的。

3. PascalCase 命名方式

与 camelCase 命名方式不同,PascalCase 命名方式的第 1 个单词的首字母是大写的。当使用 PascalCase 命名方式定义一个组件时,在引用自定义元素时 kebab-case 和 PascalCase 两种形式都可以使用,也就是说 my-component-name 和 MyComponentName 都是可接受的。

以上都是官方文档所述,因为 HTML 不区分大小写,所以不管是用 kebab-case 还是 camelCase 方式命名的组件,在写入 HTML 后都要改写为 kebab-case 形式。

【基本语法】

```
// 在组件中局部注册,自定义组件名称必须加上引号,如'kebab-cased-component'  
components: {  
  // 使用 kebab-case 注册  
  'kebab-cased-component': { /* ... */ },  
  // 使用 camelCase 注册  
  'camelCaseComponent': { /* ... */ },  
  // 使用 PascalCase 注册  
  'PascalCasedComponent': { /* ... */ }  
}
```

在 HTML 模板中使用 kebab-case 命名方式。

```
<!-- 在 HTML 模板中始终使用 kebab-case 命名方式 -->  
<kebab-cased-component></kebab-cased-component>  
<camel-cased-component></camel-cased-component>  
<pascal-cased-component></pascal-cased-component>
```

当使用字符串模式时,可以不受 HTML 大小写敏感的限制。也就是说,实际上在模板中可以使用以下方式引用这些组件:

- kebab-case;
- camelCase 或 kebab-case(如果组件已经被定义为 camelCase 形式);
- kebab-case、camelCase 或 PascalCase (如果组件已经被定义为 PascalCase 形式)。

```
// 在组件中定义  
components: {  
  'kebab-cased-component': { /* ... */ },  
  'camelCaseComponent': { /* ... */ },  
  'PascalCasedComponent': { /* ... */ }  
}  
// 在字符串模板中,可以采用以下格式引用  
const app = Vue.createApp({});  
app.component('MyCom', {  
  template: `<div>  
    <kebab-cased-component></kebab-cased-component>  
    <camel-cased-component></camel-cased-component>  
    <camelCaseComponent></camelCaseComponent>  
    <pascal-cased-component></pascal-cased-component>  
    <pascalCasedComponent></pascalCasedComponent>  
    <PascalCasedComponent></PascalCasedComponent>  
  </div>`  
})
```

5.1.2 组件注册

组件注册通常分为全局注册和局部注册。全局组件适用于所有实例,局部组件仅供本实例使用。

1. 全局注册组件

【基本语法】

```
// 全局组件注册
const app = Vue.createApp({})
app.component(tagName, options)
// 直接定义选项
app.component('myComponet', {
  // data 必须定义为函数, 并通过 return 返回相关数据
  template: 'HTML 元素定义'
})
// 也可以链式注册全局组件
app
  .component('ComponentA', ComponentA)
  .component('ComponentB', ComponentB)
  .component('ComponentC', ComponentC)
```

【语法说明】

tagName 为组件名, options 为配置选项, 选项是一个对象, 通过模板(template)选项定义组件的外观。ComponentA、ComponentB、ComponentC 为已定义 JavaScript 对象或选项对象。当模板中包含多个 HTML 元素时, 必须使用一个根元素包裹其他 HTML 元素, 否则定义的组件不会生效。与 Vue 2.x 中 Vue 实例 data 选项的定义方法不同, 在组件中定义 data 选项时, 必须定义为函数, 其中的数据通过 return 返回。也可以使用 data(){return{}} 格式来定义, 具体格式如下。

```
data: function(){          // 也可以使用 data(){return { ... }}
  return {
    //定义相关数据属性
  }
}
```

注册后, 可以调用组件, 调用方式如下。

```
<myComponet></ myComponet >
<ComponentA/>
<ComponentB/>
<ComponentC/>
```

在 Vue 组件中, 必须通过 components 选项定义自己的组件。也可以使用普通的 JavaScript 对象定义组件, 然后在全局注册和局部注册时直接引用。全局注册的组件可用于此应用程序内任何组件的模板中, 甚至适用于所有子组件, 这意味着所有组件也将彼此内部可用。

2. 局部注册组件

【基本语法】

```
const componentA = {template: '<h1> h1 - Js 自定义局部组件 - A 组件</h1>'};
// 创建 Vue 实例
const { createApp } = Vue;
const App = {          // 定义根组件
  components: {        // 局部注册子组件
    mycomp1: componentA,
    mycomp2: tmp1
  }
})
createApp(App).mount('# app');
//也可以采用以下方法
createApp({
```

```

components: {
  mycomp1: componentA,
  mycomp2: tmp1
}
}).mount('#app');

```

【语法说明】

局部注册时,需要在 components 选项下注册组件,mycomp1 和 mycomp2 为组件名,componentA 为组件对应的 JavaScript 对象或选项对象。在 Vue 中,一个组件本质上是一个拥有预定义选项的 Vue 实例,在 Vue 中注册组件非常简单。

【例 5-1】 全局注册与局部注册组件实战。代码如下,页面效果如图 5-2 所示。



图 5-2 全局注册与局部注册组件

```

1. <!-- vue-5-1.html -->
2. <!DOCTYPE html>
3. <html>
4.   <head>
5.     <meta charset = "UTF-8" />
6.     <title>组件基础</title>
7.     <script src = "../js/vue.global.js"></script>
8.     <style type = "text/css">
9.       button {border: 1px dotted blue;border-radius: 8px;
10.        width: 100px;height: 40px; margin: 5px 10px;}
11.     </style>
12.   </head>
13.   <body>
14.     <div id = "app">
15.       <fieldset>
16.         <legend>定义全局组件与复用</legend>
17.         <my-com></my-com>
18.         <my-com></my-com>
19.         <my-com></my-com>
20.       </fieldset>
21.       <fieldset>
22.         <legend>定义全局组件</legend>
23.         <mycomp2></mycomp2>
24.       </fieldset>
25.       <fieldset>
26.         <legend>定义局部组件</legend>
27.         <mycomp1></mycomp1>
28.       </fieldset>
29.     </div>
30.     <script type = "text/javascript">

```

```

31. const { createApp } = Vue;    // 解构赋值 createApp
32. // 定义 JS 对象 componentA
33. var componentA = {
34.   template: "<h1> h1 - JS 自定义局部组件 - A 组件</h1>",
35. };
36. // 定义 JS 对象 componentB
37. var componentB = {
38.   data() {
39.     return {title: "我自己定义的标题信息!"};
40.   },
41.   template: `
42.     <div>
43.       <p>多个 HTML 元素时,使用根元素 div</p>
44.       <h3 v-bind:title = "title"> h3 - JS 自定义全局组件 - B 组件,盘旋会有提示
45.     </div>`,
46.   };
47. // 局部注册组件 mycomp1
48. const App = {
49.   components: {mycomp1: componentA,},
50. };
51. const app1 = createApp(App);
52. // 全局注册组件 my-com, 定义 1 个计数器的按钮
53. app1.component("my-com", {
54.   data() {
55.     return {count: 0,};
56.   },
57.   template: '<button v-on:click = "count++">单击{{count}}次按钮</button>',
58. });
59. // 通过 JavaScript 对象 componentB 来定义 mybutton2 组件
60. app1.component("mycomp2", componentB);
61. // 定义 App 组件,并在根组件中注册子组件 mybutton1
62. app1.mount("#app");          // 挂载
63. </script>
64. </body>
65. </html>

```

上述代码中,第 14~29 行定义视图内容,主要用于展示全局注册和局部注册组件的使用;第 33~46 行定义 componentA、componentB 两个 JavaScript 组件对象供引用;第 49 行定义局部注册组件 mycomp1;第 51 行使用 createApp(App) 创建 Vue 实例 app1;第 53~58 行使用 app1.component() 定义全局注册组件 my-com,其中第 57 行模板中按钮绑定单击事件,实现 count 变量累加,并同时修改按钮的提示信息;第 60 行全局注册组件 mycomp2。

5.2 组件间通信

组件的作用域是孤立的。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。父组件可以使用 props 属性把数据传给子组件。

在 Vue 中,父子组件的关系可以总结为“props 向下传递,事件向上传递”。父组件通过 props 向子组件下发数据,子组件通过事件向父组件发送消息,如图 5-3 所示。子组件需要显式地用 defineProps() 函数声明 props。props 的值可以是两种,一种是字

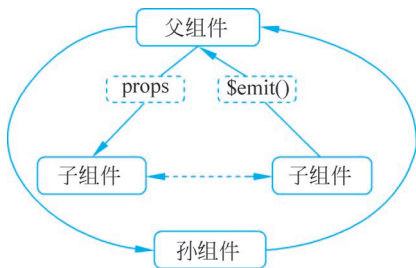


图 5-3 父子组件通信

字符串数组,另一种是对象。

5.2.1 父组件向子组件传值

1. 使用 props 传递数据

【基本语法】

```
<!-- HTML 模板部分 -->
<div id="app">
  <my-component message="父组件通过 props 传递参数"></my-component>
</div>
<!-- JS 部分 -->
<script type="text/javascript">
  const app = Vue.createApp({});
  //组件定义
  app.component('my-component',{
    props: ['message'],
    template: '<div>{{ message }}</div>'
  });
  App.mount('#app');
</script>
```

【语法说明】

`<my-component>` `</my-component>` 为自定义组件,采用 kebab-case 命名方式,相当于一个 HTML 元素。message 为组件的属性,其值作为传递给子组件的内容。props 属性为字符串数组(用方括号包围),其中的属性必须使用引号包围(如 'message'),可以在模板中通过文本插值的方式使用。这种父子组件通信是单向的。组件中的数据共有 3 种形式: data、props、computed。



注意 组件中 data 和 props 都可以为组件提供数据,但它们是有区别的。data 选项的类型为对象,对象中返回的数据属于组件内部数据,只能用于组件本身。props 选项的类型可以是字符串数组,也可以是对象,用来声明组件从外部接收的数据。这两种数据都可以在模板(template)、计算属性(computed)和方法(methods)中使用。

【例 5-2】 父组件通过 props 向子组件下发数据实战。代码如下,页面效果如图 5-4 所示。



图 5-4 父组件通过 props 下发数据

```
1. <!-- vue-5-2.html -->
2. <!DOCTYPE html>
3. <html>
4.   <head>
```

```
5. <meta charset = "UTF - 8" />
6. <script src = "../js/vue.global.js"></script>
7. <title> props 下发数据</title>
8. </head>
9. <body>
10. <div id = "app">
11. <my-component message = "父组件 props 下发数据"></my-component>
12. </div>
13. <script type = "text/javascript">
14.   const { createApp } = Vue;
15.   // 创建 Vue 实例
16.   const app = createApp({});
17.   // 全局注册组件 my-component
18.   app.component("my-component", {
19.     data: function() {
20.       return {
21.         title: "我是组件中的内部数据!",
22.       };
23.     },
24.     props: ["message"],
25.     template: "<div>{{ message }} - <strong>{{title}}</strong></div>",
26.   });
27.   app.mount("#app");
28. </script>
29. </body>
30. </html>
```

2. 静态 props 传递数据

HTML 中的属性名是大小写不敏感的,所以浏览器会把所有大写字符解释为小写字符。这意味着当使用 DOM 中的模板时,camelCase 形式的 props 名称需要使用其等价的 kebab-case 形式。

通常父组件单向(正向)传递数据给子组件,需要以下 3 个步骤。

- (1) 创建父、子组件构造器。通过 `Vue.extend()` 方法构建组件。
- (2) 注册父、子组件。可以全局或局部注册各类组件。
- (3) 在 Vue 实例范围内使用组件。

【例 5-3】 静态 props 传递数据实战。代码如下,页面效果如图 5-5 所示。



图 5-5 静态 props 传递数据

```
1. <!-- vue - 5 - 3.html -->
2. <!DOCTYPE html>
3. <html>
4. <head>
5. <meta charset = "UTF - 8" />
```

```
6.     <title>props - 父组件将数据传递给子组件</title>
7.     <script type="text/javascript" src="../js/vue.global.js"></script>
8.   </head>
9.   <body>
10.    <!-- # app 是 Vue 实例挂载的元素,在挂载元素范围内使用组件 -->
11.    <div id="app">
12.      <h3>我的待办事项 - 日期:{{today}}</h3>
13.      <mytodo :todo="data" data="myTodos"></mytodo>
14.    </div>
15.  </body>
16.  <script>
17.    // 定义根组件 App,准备 data
18.    const App = {
19.      data() {
20.        return {
21.          today: new Date().toLocaleDateString(),
22.          myTodos: [
23.            { id: 0, text: "完成月度工作总结" },
24.            { id: 1, text: "主持本周项目讨论会" },
25.            { id: 2, text: "通知周五提交周工作计划" },
26.          ],
27.        };
28.      },
29.    };
30.    // 创建 Vue 实例
31.    const app = Vue.createApp(App);
32.    // 构建一个子组件,使用字符串模板(反引号表示)
33.    const todoChild = {
34.      template: `<li>{{ text }}</li>`,
35.      props: {
36.        text: { type: String, default: "" },
37.      },
38.    };
39.    // 构建一个父组件,使用字符串模板(反引号表示)
40.    const todoParent = {
41.      template: `
42.        <ul>
43.          <todo-item v-for="(item, index) in todoData" v-text="item.text" v-bind:
key="item.id"></todo-item>
44.        </ul>
45.      `,
46.      props: { todoData: { type: Array, default: [] }, },
47.      components: { todoItem: todoChild }, // 注册局部子组件
48.    };
49.    // 注册全局组件 mytodo
50.    app.component("mytodo", todoParent);
51.    app.mount("#app");
52.  </script>
53. </html>
```

上述代码中,第 18~29 行定义根组件 App,并设置 data 选项,为 myTodos 对象数组准备数据;第 33~48 行定义两个组件,分别为 todoChild 和 todoParent,并在 todoParent 组件中局部注册子组件 todoChild 为 todoItem;第 50 行将 todoParent 组件注册为全局组件 mytodo;第 51 行进行挂载。

3. 动态 props 传递数据

要动态地绑定父组件的数据到子模板的 props,与绑定到任何普通的 HTML 属性相类似,就是使用 v-bind 指令。每当父组件的数据发生变化时,都会实时地将变化后的数据传递

给子组件。

【例 5-4】 动态 props 传递数据实战。代码如下,页面效果如图 5-6 和图 5-7 所示。

```
1. <!-- vue-5-4.html -->
2. <!DOCTYPE html>
3. <html>
4.   <head>
5.     <meta charset = "UTF-8" />
6.     <title> props - 父组件将动态传递数据</title>
7.     <script type = "text/javascript" src = "../js/vue.global.js"></script>
8.   </head>
9.   <body>
10.    <div id = "app">
11.      <h3> 1. 父组件动态 props 传递数据给子组件(v-model)</h3>
12.      <input type = "text" v-model = "parentInputText" placeholder = "请输入内容" />
13.      <h3> 2. 子组件接收动态数据(v-bind)</h3>
14.      <my-component :message = "parentInputText"></my-component>
15.      <h3> 3. 子组件修改其值(computed),但不影响父组件数据</h3>
16.      <my-component :message = "changeParentInputText"></my-component>
17.      <h3> 4. 子组件使用 v-model 绑定值时控制台会报错,也不会改变父组件的数据</h3>
18.      <my-component-1 :message = "parentInputText"></my-component-1>
19.    </div>
20.  </body>
21.  <script>
22.    //定义根组件,并准备数据
23.    const App = {
24.      data() {
25.        return { parentInputText: "", };
26.      },
27.      computed: {
28.        //通过计算属性修改子组件数据
29.        changeParentInputText: function() {
30.          return "子组件修改其值了!" + this.parentInputText;
31.        },
32.      },
33.    };
34.    const app = Vue.createApp(App);
35.    //全局注册组件 my-component,组件内直接使用 message
36.    app.component("my-component", {
37.      props: ["message"],
38.      template: "<div>{{ message }}</div>",
39.    });
40.    //全局注册组件 my-component-1,组件内通过 v-model 指令绑定 message
41.    app.component("my-component-1", {
42.      props: ["message"],
43.      template: '<div><input type = "text" v-model = "message" /></div>',
44.    });
45.    app.mount("#app"); // 挂载视图
46.  </script>
47. </html>
```

上述代码中,第 18 行引用 my-component-1 组件,此时在组件中修改 props 传递过来的数据,控制台会发出警告,如图 5-7 所示。所以,通常采用 computed 计算属性修改 props 传递过来的值。本地定义属性,并将 props 作为初始值,props 传入之后需要进行转换。



注意 在使用 props 传值时,如果不使用 v-bind 指令传递数字、布尔、数组、对象类型的数据,此时传递的数据都是字符串类型,由于模板中未使用 v-bind 指令绑定属性,因此不会被编译。



图 5-6 动态 props 传递数据

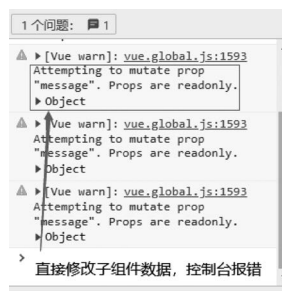


图 5-7 修改数据控制台报错

【例 5-5】传值时 v-bind 的使用实战。代码如下,页面效果如图 5-8 所示。



图 5-8 传值时 v-bind 的使用

```

1. <!-- vue-5-5.html -->
2. <!DOCTYPE html>
3. <html>
4.   <head>
5.     <meta charset="UTF-8"/>
6.     <script src="../../js/vue.global.js"></script>
7.     <title>传值时 v-bind 的使用</title>
8.   </head>
9.   <body>
10.    <div id="app">
11.      <h3>使用 v-bind 时,将['123','456','789','ABD']作为数组类型</h3>
12.      <my-component: message="[ '123', '456', '789', 'ABD' ]"></my-component>
13.      <h3>未使用 v-bind 时,将['123','456','789','ABD']作为字符串</h3>
14.      <my-component message="[ '123', '456', '789', 'ABD' ]"></my-component>
15.    </div>
16.    <script type="text/javascript">
17.      const app = Vue.createApp({});
18.      app.component("my-component", {
19.        props: ["message"],
20.        template: "<div> message.length = {{ message.length }}</div>",
21.      });
22.      app.mount("#app");
23.    </script>
24.  </body>
25. </html>

```

4. props 数据验证

组件 props 选项的值可以是数组类型,也可以是对象类型。props 选项的对象类型可以用于对外部传递进来的参数进行数据验证。当封装了一个组件时,对于内部接收的参数进行校

验是非常必要的。部分代码如下。

```
<script>
const app = Vue.createApp({});
app.component('my-comp',{
  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,而且是必选
    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 {};
      }
    },
    //自定义验证函数
    propF: {
      validator: function(value) {
        return value > 10;
      }
    }
  }
});
</script>
```

其中,default 表示默认值;required 表示必选。

验证的类型(type)可以是 String、Number、Boolean、Object、Array、Function。type 也可以是一个自定义构造器,使用 instanceof 检测。props 验证失败时,开发版本下会在控制台抛出一条警告。

【例 5-6】 props 数据验证实战。代码如下,页面效果如图 5-9 和图 5-10 所示。

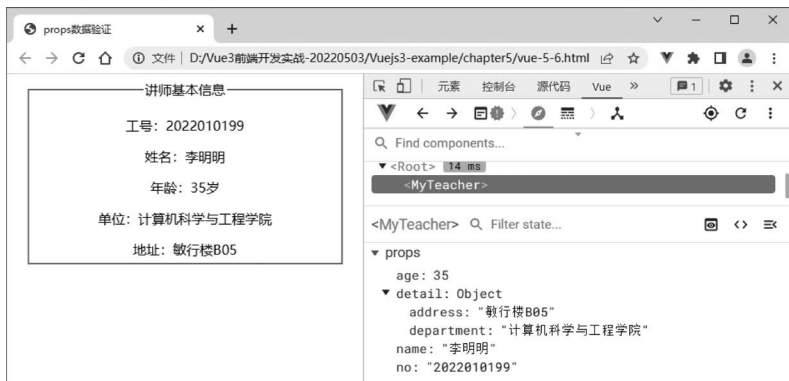


图 5-9 props 数据验证

```

1. <!-- vue-5-6.html -->
2. <!DOCTYPE html>
3. <html>
4.   <head>
5.     <meta charset = "UTF-8" />
6.     <title>props 数据验证</title>
7.     <script src = "../js/vue.global.js"></script>
8.     <style type = "text/css">
9.       fieldset {width: 350px;height: 200px;margin: 0 auto;text-align: center;}
10.    </style>
11.  </head>
12.  <body>
13.    <div id = "app">
14.      <fieldset>
15.        <legend align = "center">讲师基本信息</legend>
16.        <my-teacher name = "李明明" no = "2022010199" :age = "35"></my-teacher>
17.      </fieldset>
18.    </div>
19.    <script>
20.      const app = Vue.createApp({});
21.      // 全局注册组件 my-teacher
22.      app.component("my-teacher", {
23.        props: {
24.          no: {
25.            type: String,
26.            required: true,                //必选,不选报错
27.          },
28.          name: {
29.            type: String,
30.            required: true,                //必选,不选报错
31.          },
32.          age: {
33.            type: Number,
34.            validator: function(value) {
35.              return value >= 0 && value <= 130;    //年龄为 0~130
36.            },
37.          },
38.          detail: {
39.            type: Object,
40.            default: function() {
41.              return {
42.                department: "计算机科学与工程学院",
43.                address: "敏行楼 B05",
44.              };
45.            },
46.          },
47.        },
48.        template: `
49.          <div class = "person">
50.            <p>工号:{{this.no}}</p>
51.            <p>姓名:{{this.name}}</p>
52.            <p>年龄:{{this.age}}岁</p>
53.            <p>单位:{{this.detail.department}}</p>
54.            <p>地址:{{this.detail.address}}</p>
55.          </div>
56.        `,
57.      });
58.      app.mount("#app");
59.    </script>
60.  </body>
61. </html>

```

上述代码中,第 16 行使用 my-teacher 组件设置 name 和 no 两个必选属性以及 age 属性并赋值。第 22~57 行全局注册组件 my-teacher,定义 props 为对象,其中定义 no、name 为字符串,必选;age 为数字型,取值范围为 0~130;detail 为对象,包含 address 和 department 两个属性。

将第 16 行代码中的 no 属性设置项取消,同时将“:age=35”改为“:age=135”,将会报错,如图 5-10 所示。

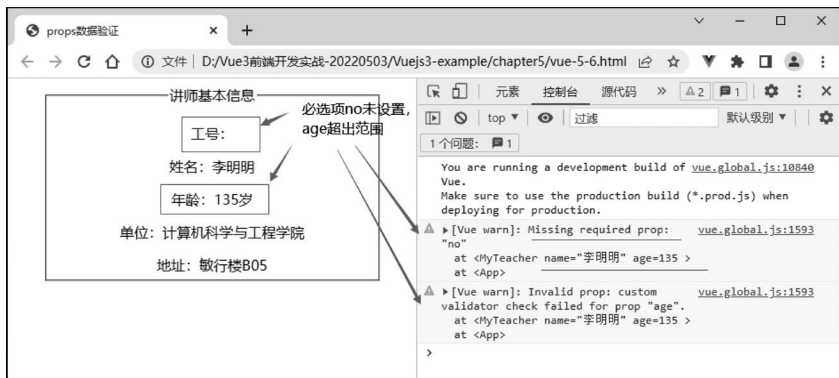


图 5-10 组件属性设置不符合验证要求时报错

5.2.2 子组件向父组件传值

1. 自定义事件

当子组件需要向父组件传递数据时,需要使用自定义事件。子组件使用 `$emit()` 触发自定义事件,父组件使用 `$on()` 侦听子组件的事件。父组件也可以直接在子组件的自定义标记上使用 `v-on` 指令侦听子组件触发的自定义事件,数据就通过自定义事件进行传递。

1) “父组件 v-on: 自定义事件”传递数据

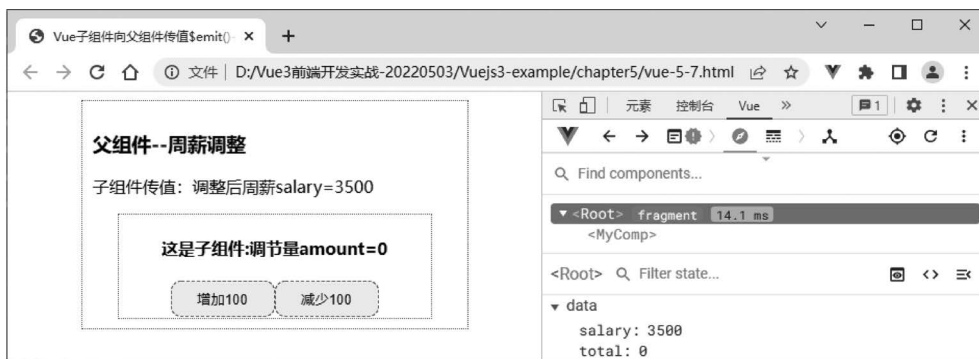
```
<child-comp v-on: eventName = "functionName"></child-comp>
//在父组件的 Vue 组件中定义 methods 选项
methods:{
  functionName(postdata){
    //对传递来的数据 postdata 进行处理,并对父组件的数据进行运算
  }
}
```

2) 子组件 `$emit()` 触发自定义事件传递数据

```
<button @click = 'increase'>增加 100 </button>
methods: {
  'increase': function() {
    this.$emit('eventName', data); // 第 1 个参数为自定义事件名称,第 2 个参数为数据
  },
  ...
}
```

子组件通过 `this.$emit('eventName',data)` 方式触发父组件约定的事件绑定的处理函数 `functionName`,同时子组件传递数据给父组件。父组件使用 `v-on` 指令侦听自定义事件。需要注意, `$emit('eventName', data)` 函数的第 1 个参数是自定义事件的名称,第 2 个参数为数据,数据可以有多个。

【例 5-7】 子组件向父组件传值实战——周薪调整。代码如下,页面效果如图 5-11 所示。



(a) 初始页面



(b) 单击“增加100”按钮



(c) 单击“减少100”按钮

图 5-11 子组件向父组件传值(1)

```

1.  <!-- vue-5-7.html -->
2.  <!DOCTYPE html>
3.  <html>
4.    <head>
5.      <meta charset="UTF-8" />
6.      <title>Vue 子组件向父组件传值 $emit() + v-on</title>
7.      <script src="../js/vue.global.js"></script>
8.      <style type="text/css">
9.        button {width: 100px;height: 35px;border: 1px dashed red;border-radius: 10px;}
10.       .childClass {margin: 0 auto; width: 300px;height: 100px;
11.         border: 1px dotted green;text-align: center;}
12.       #app {margin: 0 auto;width: 350px;height: 200px;
13.        border: 1px dotted green; padding: 10px; }
14.     </style>
15. </head>
16. <body>
17.   <div id="app">
18.     <h3>父组件--周薪调整</h3>
19.     <p>子组件传值:调整后周薪 salary={{salary+total}}</p>
20.     <my-comp @increase="changeTotal" @reduce="changeTotal"></my-comp>
21.   </div>
22.   <script>
23.     // 1. 定义根组件 App,并配置 data 和 methods 选项
24.     const App = {
25.       data() {
26.         return {
27.           total: 0,           //父组件的数据
28.           salary: 3500,
29.         };
30.       },

```

```

31.     methods: {
32.         changeTotal(total) {
33.             this.total = total; //total 是子组件传过来的数据
34.         },
35.     },
36. };
37. // 2. 以 App 组件创建 Vue 实例
38. const app = Vue.createApp(App);
39. // 3. 全局注册组件 my-comp
40. app.component("my-comp", {
41.     template: `
42.         <div class = 'childClass'>
43.             <h4>这是子组件:调节量 amount = {{amount}}</h4>
44.             <button @click = 'increase'>增加 100 </button>
45.             <button @click = 'reduce'>减少 100 </button>
46.         </div> `,
47.     data() {
48.         // 子组件的数据
49.         return { amount: 0 };
50.     },
51.     methods: {
52.         // 子组件通过事件处理函数执行 $ emit(),传递数据
53.         increase() {
54.             // 调节量增 100
55.             this.amount += 100;
56.             this.$ emit("increase", this.amount);
57.         },
58.         reduce() {
59.             // 调节量减 100
60.             this.amount -= 100;
61.             this.$ emit("reduce", this.amount);
62.         },
63.     },
64. });
65. // 挂载视图并渲染
66. app.mount("# app");
67. </script>
68. </body>
69. </html>

```

上述代码中,第 17~21 行定义视图,在视图中使用子组件< my-comp >侦听自定义事件 increase 和 reduce,当事件发生时调用 changeTotal 方法实现传值;第 40~64 行定义子组件,在子组件< template >标记内定义两个按钮,并定义单击事件处理子组件的数据,将子组件的数据传出。

2. 使用 v-model 指令传递数据

由于 Vue 3.x 中 v-model 指令的使用方法已经变更,用于自定义组件时,v-model 指令中 prop 和事件默认名称已更改为以下格式。

```

prop:value -> modelValue;
event:input -> update:modelValue;

```

在父组件上使用 v-model:modelValue='xxx'指令,子组件中使用 this.\$ emit('update:modelValue',this.子组件属性)进行传值。

1) 父组件中通过 v-model 指令绑定数据

```

<div id = "app">
  <my-comp v-model: modelValue = modelValue '></my-comp>
</div>

```

2) 子组件中通过事件触发 \$emit('update:modelValue', data)

```
const app = Vue.createApp(App);
app.component('my-comp', {
  template: `
    <div class = 'childClass'>
      <button @click = 'increase'>增加 1 </button>
    </div>`,
  methods: {
    increase() { //计数器增 1
      this.counter++;
      this.$emit('update:modelValue', this.counter);
    },
  }
})
```

在 Vue 2.x 中,在组件上使用 v-model 指令相当于绑定 value prop 和 input 事件。

```
<ChildComponent v-model = "pageTitle" />
<ChildComponent :value = "pageTitle" @input = "pageTitle = $event.target.value" />
```

子组件中的事件处理函数中,使用 \$emit() 发送数据。

```
this.$emit("input", postData);
```

【例 5-8】 v-model 指令传递数据实战——课程刷分。代码如下,页面效果如图 5-12 所示。

```
1. <!-- vue-5-8.html -->
2. <!DOCTYPE html>
3. <html>
4.   <head>
5.     <meta charset = "UTF-8" />
6.     <title>使用 v-model 动态传递数据</title>
7.     <script src = "../js/vue.global.js"></script>
8.     <style type = "text/css">
9.       button {width: 100px; height: 35px;
10.        border: 1px dashed red; border-radius: 10px; }
11.       .childClass {width: 280px; height: 120px; margin: 0 auto;
12.        border: 1px dotted green; text-align: center; }
13.       #app {margin: 0 auto; width: 350px; height: 250px;
14.        border: 1px dotted green; padding: 10px;}
15.     </style>
16.   </head>
17.   <body>
18.     <div id = "app">
19.       <h3>父组件 v-model: 课程刷分</h3>
20.       <p v-once>课程初始成绩: {{score}}</p>
21.       <p>子组件传值: 调整后课程成绩 score = {{score + total}}</p>
22.       <my-comp v-model: total = "total"></my-comp>
23.     </div>
24.     <script>
25.       const App = {
26.         data() {
27.           return {           // 父组件的数据
28.             score: 65,
29.             total: 0,
30.           };
31.         },
32.       };
33.       const app = Vue.createApp(App);
34.       // 全局注册组件 my-comp
35.       app.component("my-comp", {
```

```

36.     template: `
37.         <div class = 'childClass'>
38.             <h4>这是子组件:刷分量 amount = {{amount}}</h4>
39.             <button @click = 'increase'>增加 1</button>
40.             <button @click = 'reduce'>减少 1</button>
41.         </div> `,
42.     data() {
43.         //子组件的数据
44.         return { amount: 0 };
45.     },
46.     methods: {
47.         increase() {
48.             //刷分量增 1
49.             this.amount++;
50.             this.$ emit("update:total", this.amount);
51.         },
52.         reduce() {
53.             //刷分量减 1
54.             this.amount--;
55.             this.$ emit("update:total", this.amount);
56.         },
57.     },
58.   });
59.   app.mount("# app");
60. </script>
61. </body>
62. </html>

```



(a) 初始页面



(b) 单击“增加1”按钮



(c) 单击“减少1”按钮

图 5-12 子组件向父组件传值(2)



注意 虽然 Vue 允许子组件修改父组件数据,但在实际业务中,子组件应该尽量避免依赖父组件的数据,更不应该主动修改它的数据。由于父、子组件属于紧耦合,如果仅从父组件来看,很难理解父组件的状态,因为它可能被任意组件修改。所以,通常情况下,组件能修改自己的状态,父、子组件之间最好还是通过 props 和 \$emit() 通信。

5.2.3 父链与子组件索引

除了上述方法外,采用父链与子组件索引同样可以实现组件间通信。

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(返回值为数组类型)通过索引值访问它的所有子组件,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。

子组件利用父链 this.\$parent 获取设置父组件的数据,部分代码如下。

```
const msg = this.$parent.message;           //获取数据
this.$parent.message = '消息:组件 my-comp 修改数据' //设置父组件的数据
```

子组件索引 this.\$refs.indexName 修改组件数据,部分代码如下。

```
<my-comp1 ref="comp1"></my-comp1>
<my-comp2 ref="comp2"></my-comp2>
// 在事件处理函数中或 JS 代码中
this.message = this.$refs.comp1.message;           //通过 $refs 获取子组件的数据
this.$refs.comp1.message = this.message            //通过 $refs 设置子组件的数据
```

【例 5-9】 父链与子组件索引应用实战。代码如下,页面效果如图 5-13 和图 5-14 所示。

```
1. <!-- vue-5-9.html -->
2. <!DOCTYPE html>
3. <html lang="en">
4.   <head>
5.     <meta charset="UTF-8" />
6.     <title>父链与子组件索引的应用</title>
7.     <script src="../../js/vue.global.js"></script>
8.   </head>
9.   <body>
10.    <div id="app">
11.      <fieldset>
12.        <legend>子组件利用父链修改父组件数据</legend>
13.        <p>{{message}}</p>
14.        <my-comp></my-comp>
15.      </fieldset>
16.      <fieldset>
17.        <legend>父组件通过子组件索引访问子组件数据</legend>
18.        <p v-show="message1!= '' || message2!= ''">{{message1}}, {{message2}}</p>
19.        <my-comp1 ref="comp1"></my-comp1>
20.        <my-comp2 ref="comp2"></my-comp2>
21.        <button @click="handlerCompRef">通过 ref 获取子组件实例</button>
22.      </fieldset>
23.    </div>
24.    <script>
25.      const App = {
26.        data() {
27.          return { message: "", message1: "", message2: "" };
28.        },
29.        methods: {
30.          handlerCompRef: function() {
```

```
31.      // 通过 $ refs 获取子组件实例
32.      this.message1 = this.$refs.comp1.message;
33.      this.message2 = this.$refs.comp2.message;
34.    },
35.  },
36.  };
37.  const app = Vue.createApp(App);
38.  // 全局注册组件 my-comp
39.  app.component("my-comp", {
40.    template:
41.      '<button @click="changeMessage">通过父链直接修改数据</button>',
42.    methods: {
43.      changeMessage: function() {
44.        // 通过 this.$parent 直接修改父组件的数据
45.        this.$parent.message = "消息:组件 my-comp 修改数据";
46.      },
47.    },
48.  });
49.  // 全局注册组件 my-comp1
50.  app.component("my-comp1", {
51.    template: "<div>子组件 1 - {{message}}</div>",
52.    data() {
53.      return { message: "my-comp1 - 欢迎使用我的数据!" };
54.    },
55.  });
56.  // 全局注册组件 my-comp2
57.  app.component("my-comp2", {
58.    template: "<div>子组件 2 - {{message}}</div>",
59.    data() {
60.      return { message: "my-comp2 - 请下载使用优质资源!" };
61.    },
62.  });
63.  app.mount("#app");
64. </script>
65. </body>
66. </html>
```

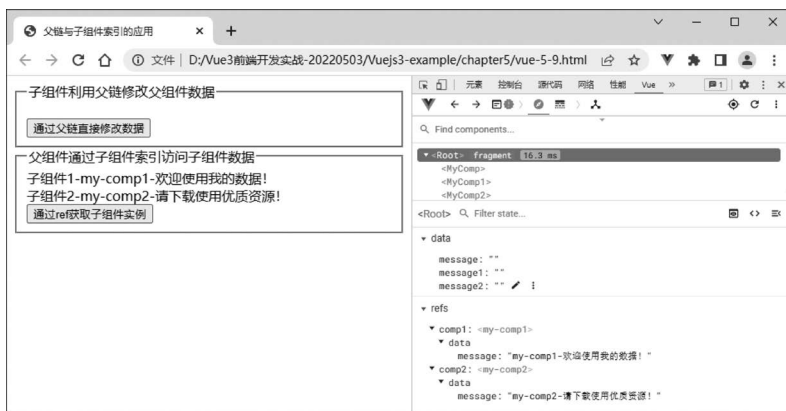


图 5-13 父链与子组件索引应用初始页面

上述代码中,第 19 行和第 20 行为两个组件设置 ref 属性,值分别为 comp1 和 comp2;第 32 行和第 33 行父组件通过 this.\$refs 访问指定名称的子组件的数据;第 45 行子组件通过 \$parent 获取或设置父组件数据。



图 5-14 单击按钮后数据获取页面效果



注意 `$refs` 只在组件渲染完成后才填充,并且它是非响应式的。它仅仅作为一个直接访问子组件的应急方案,应当尽量避免在模板或计算属性中使用 `$refs`。

5.3 插槽

在开发组件时,组件内一些子元素希望是由调用者来定义,组件只负责核心功能,其他非核心由用户自由定义,可以增加组件的灵活性和可扩展性。这种场景非常适合使用插槽。插槽(Slot)是 Vue 提出的一个概念,用于决定将所携带的内容插入指定的某个位置,从而使模板分块,具有模块化的特点和更强的重用性。

插槽是否显示、怎样显示是由父组件控制的,而插槽在哪里显示就由子组件进行控制,父页面在组件标记内插入任意内容,子组件内插槽控制摆放位置(匿名插槽、具名插槽)。

插槽分类如下。

- (1) 匿名插槽: 默认(default)插槽。没有命名,有且只有一个。
- (2) 具名插槽: 指 `<slot>` 标记带 `name` 属性的插槽。
- (3) 作用域插槽: 子组件内数据可以被父页面拿到(解决了数据只能从父页面传递给子组件的问题)。

5.3.1 匿名插槽

匿名插槽使用 `<slot>` `</slot>` 标记为需要传递的内容占位,类似于内容占位符。

【基础语法】

- (1) 在父组件中使用子组件标记,并携带传递的内容。

```
<child><p>父组件需要分发给子组件的内容...</p></child>
```

- (2) 在子组件模板中插入插槽标记。

```
const Comp1 = {
  template: `
    <div class="comp">
      <h3>这是子组件...</h3>
      <slot>插槽默认的内容 1</slot>
      <slot>插槽默认的内容 2</slot>
    </div>
  `
}
```

```

    </div>
  },
  //App 组件内注册子组件
  const App = {
    components: {child: Comp1,}
  }

```

【语法说明】

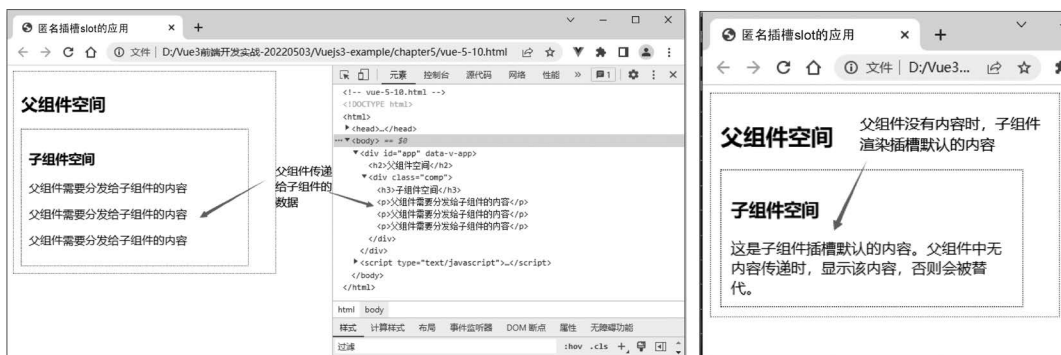
子组件渲染时,会将< slot >标记用父组件中子组件标记的内容替换。子组件中如果有多个匿名插槽,将会同时被父组件传递的内容所替换。如果在子组件的模板中没有插入< slot >标记,则父组件中使用子组件标记的内容将被忽略。

【例 5-10】 匿名插槽实战。代码如下,页面效果如图 5-15 所示。

```

1.  <!-- vue-5-10.html -->
2.  <!DOCTYPE html>
3.  <html>
4.    <head>
5.      <meta charset = "UTF-8" />
6.      <title>匿名插槽 slot 的应用</title>
7.      <script type = "text/javascript" src = "../js/vue.global.js"></script>
8.      <style type = "text/css">
9.        .comp {width: 300px;border: 1px dotted black; padding: 10px;}
10.       #app {border: 1px dotted green;width: 350px;padding: 10px;}
11.      </style>
12.    </head>
13.    <body>
14.      <div id = "app">
15.        <h2>父组件空间</h2>
16.        <child>
17.          <p>父组件需要分发给子组件的内容</p>
18.        </child>
19.      </div>
20.      <script type = "text/javascript">
21.        const Comp1 = {
22.          template: `
23.            <div class = "comp">
24.              <h3>子组件空间</h3>
25.              <slot></slot>
26.              <slot></slot>
27.              <slot>这是子组件插槽默认的内容.父组件中无内容传递时,显示该内容,否则会被替代.</slot>
28.            </div>
29.          `,
30.        };
31.        const App = {
32.          components: {
33.            child: Comp1,
34.          },
35.        };
36.        const app = Vue.createApp(App);
37.        app.mount("#app");
38.      </script>
39.    </body>
40.  </html>

```



(a) 父组件有传递内容

(b) 父组件无传递内容，子组件有内容

图 5-15 匿名插槽的应用

5.3.2 具名插槽

slot 元素可以用一个特殊的 name 属性配置如何分发内容,多个插槽可以有不同的名字,根据具名插槽的 name 属性进行匹配,显示内容。如果有匿名插槽,那么没有匹配到的内容将会显示到匿名插槽中;如果没有匿名插槽,那么没有匹配到的内容将会被抛弃。

在向具名插槽提供内容时,也可以在一个 template 元素上使用 v-slot 指令,并以 v-slot 的参数形式提供其名称(如 v-slot:slotname)。template 元素中的所有内容都将会被传入相应的插槽中。任何没有被包裹在带有 v-slot 的 template 元素中的内容都会被视为匿名插槽的内容。当然,也可以在一个 template 元素中包裹匿名(默认)插槽的内容。

【基本语法】

(1) 在父组件中的子组件标记内,使用带 slot 属性的标记或带 v-slot 指令参数的 template 元素携带传递的内容。

```
<child>
  <p>该内容将显示在匿名插槽中.</p>
  <template>
    <p>使用匿名插槽 -- 该内容将显示在匿名插槽中.</p>
  </template>
  <template v-slot:slot1 | #slot1>
    <p>使用 v-slot 指令 -- 该内容将显示在具名插槽 slot1 中.</p>
  </template>
  <p slot='slot2'>使用 slot 属性 -- 该内容将显示在具名插槽 slot2 中.</p>
</child>
```

具名插槽可以采用缩写的方式。与 v-on 和 v-bind 一样,v-slot 也有缩写,即把参数之前的所有内容(v-slot:)替换为字符 #。例如,v-slot:slot1 可以简写为 #slot1。

(2) 在子组件模板中插入具名插槽标记(使用 name 属性)。

```
const Comp1 = {
  template: `
    <div class="comp">
      <h3>这是子组件...</h3>
      <slot name='slot1'></slot>
      <slot name='slot2'></slot>
      <slot></slot>
    </div>
  `
}
```

```
// App 组件中注册子组件
components: {child: Comp1,}
```

【语法说明】

在父组件中使用子组件标记,并在其中插入多个带 slot 属性的标记,其属性值为子组件中定义 slot 元素所指定的 name 属性值。子组件渲染时,匹配到 slot 元素(name 的属性值)会被父组件中子组件标记内具有相应的 slot 属性值的标记的内容所替换。在父组件中可以使用带 v-slot:slotname 参数的 template 元素携带传递内容,同样会匹配到带 name 属性的具名插槽。

 **注意** v-slot 一般添加在<template></template>标记上,这一点和已经废弃的 slot 属性不同。

【例 5-11】 具名插槽实战。代码如下,页面效果如图 5-16 所示。

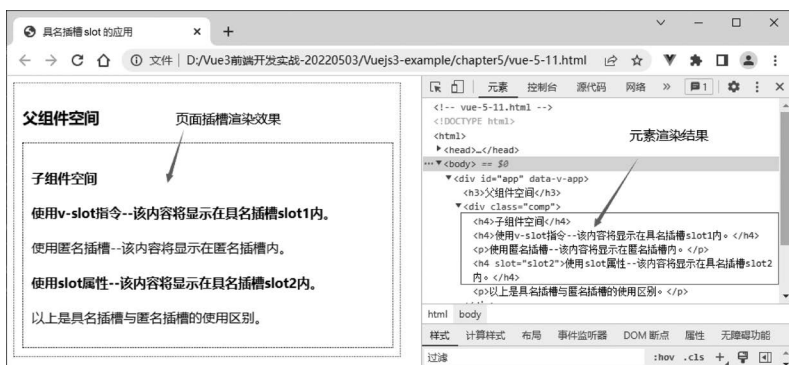


图 5-16 具名插槽

```
1. <!-- vue - 5 - 11. html -->
2. <!DOCTYPE html>
3. <html>
4.   <head>
5.     <meta charset = "UTF - 8" />
6.     <title>具名插槽 slot 的应用</title>
7.     <script type = "text/javascript" src = "../js/vue.global.js"></script>
8.     <style type = "text/css">
9.       .comp {width: 430px;border: 1px dotted black;padding: 10px;}
10.      #app {border: 1px dotted green;width: 450px;padding: 10px;}
11.    </style>
12.  </head>
13.  <body>
14.    <div id = "app">
15.      <h3>父组件空间</h3>
16.      <child>
17.        <p>使用匿名插槽 -- 该内容将显示在匿名插槽内.</p>
18.        <template v - slot:slot1>
19.          <h4>使用 v - slot 指令 -- 该内容将显示在具名插槽 slot1 内.</h4>
20.        </template>
21.        <h4 slot = "slot2">使用 slot 属性 -- 该内容将显示在具名插槽 slot2 内.</h4>
22.      </child>
23.    </div>
24.    <script type = "text/javascript">
25.      // 子组件中设有 3 个 slot,其中两个为具名插槽,一个为匿名插槽
26.      const Comp1 = {
27.        template: `
28.          <div class = "comp">
```

```

29.         <h4>子组件空间</h4>
30.         <slot name = 'slot1'></slot>
31.         <slot name = 'slot2'></slot>
32.         <slot></slot>
33.         <p>以上是具名插槽与匿名插槽的使用区别.</p>
34.     </div>
35.     `,
36.     };
37.     const App = {
38.         components: { child: Comp1 },
39.     };
40.     const app = Vue.createApp(App);
41.     app.mount("#app");
42. </script>
43. </body>
44. </html>

```

上述代码中,第 14~23 行定义父组件,并在父组件中使用子组件 child,在子组件标记内添加<p>、<template>、<h3>等标记,其中<p>标记用于匹配匿名插槽,<template>和<h3>标记用于匹配两个不同的具名插槽;第 26~36 行定义子组件 Comp1,并在 App 组件中定义 components 选项,局部注册子组件 child(第 38 行)。在子组件 Comp1 中使用两个具名插槽和一个匿名插槽。

5.3.3 作用域插槽

Vue 2.6.0 引入 v-slot 指令,提供支持 slot 和 slot-scope 属性的 API 替代方案^①。在接下来所有 2.x 版本中 slot 和 slot-scope 属性仍会被支持,但已经被官方废弃且不会出现在 Vue 3.0 中。作用域插槽可用作一个能被传递数据的可重用模板。

匿名插槽和具名插槽的内容和样式皆由父组件决定,即显示什么内容和怎样显示都由父组件决定;作用域插槽的样式由父组件决定,内容却由子组件控制。简单来说,匿名插槽和具名插槽不能绑定数据,作用域插槽是一个带绑定数据的插槽。

获取子组件 slot 中携带的数据的关键步骤如下。

(1) 在 slot 元素上使用 v-bind 指令绑定一个特定属性,这个属性称为插槽 prop。

```
<slot v-bind: customAttribute = "childDataOptions"></slot>
```

其中,customAttribute 为用户自定义的属性,为父组件提供数据对象;childDataOptions 为子组件 data 选项中的属性或对象。

(2) 在父组件访问绑定到 slot 插槽上的 prop 对象。

```

<template v-slot:default|slotname = "slotProps">
  <p>来自父组件的内容</p>
  <p>{{slotProps. customAttribute }}</p>
</template>

```

使用 v-slot 指令时可以绑定相应的具名插槽或匿名插槽(默认名称为 default,也可以省略)。slotProps 为子组件上绑定的数据(插槽的 prop 对象)。如果自定义属性 customAttribute 是对象,可以通过 slotProps. customAttribute. xxx 使用绑定的数据。

【例 5-12】 作用域插槽实战。代码如下,页面效果如图 5-17 所示。

^① <https://github.com/vuejs/rfcs/blob/master/active-rfcs/0001-new-slot-syntax.md>

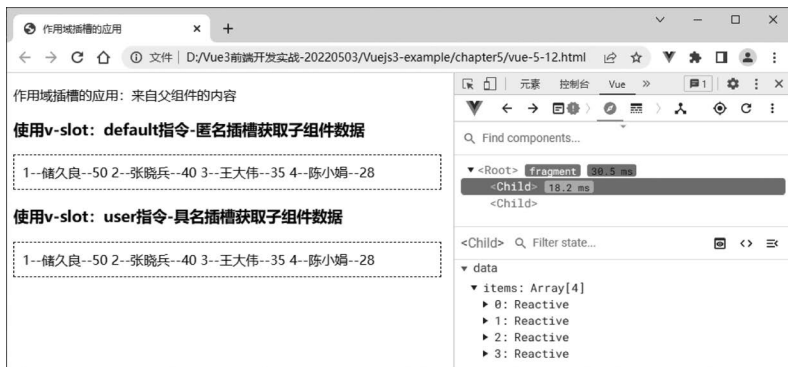


图 5-17 作用域插槽

```

1. <!-- vue-5-12.html -->
2. <!DOCTYPE html>
3. <html lang="en">
4.   <head>
5.     <meta charset="UTF-8" />
6.     <title>作用域插槽的应用</title>
7.     <script type="text/javascript" src="../../js/vue.global.js"></script>
8.     <style type="text/css">
9.       .mylist {width: 500px;border: 1px dashed black;padding: 10px;}
10.    </style>
11.  </head>
12.  <body>
13.    <div id="app">
14.      <p>作用域插槽的应用:来自父组件的内容</p>
15.      <h3>使用 v-slot:default 指令 - 匿名插槽获取子组件数据</h3>
16.      <child>
17.        <template v-slot:default="props">
18.          <span>{{props.item.id}} -- {{props.item.name}} -- {{props.item.age}}&nbsp;&nbsp;&nbsp;</span>
19.        </template>
20.      </child>
21.      <h3>使用 v-slot:user 指令 - 具名插槽获取子组件数据</h3>
22.      <child>
23.        <template v-slot:user="props">
24.          <span>{{props.item.id}} -- {{props.item.name}} -- {{props.item.age}}&nbsp;&nbsp;&nbsp;</span>
25.        </template>
26.      </child>
27.    </div>
28.    <script>
29.      // 创建 Vue 实例
30.      const app = Vue.createApp({});
31.      app.component("child", {
32.        data() {
33.          return {
34.            items: [
35.              { id: 1, name: "储久良", age: 50 },
36.              { id: 2, name: "张晓兵", age: 40 },
37.              { id: 3, name: "王大伟", age: 35 },
38.              { id: 4, name: "陈小娟", age: 28 },
39.            ],
40.          };
41.        },
42.        template: `
43.          <div class='mylist'>

```

```

44.         <slot v-for = 'item in items':item = 'item'></slot>
45.         <slot name = 'user' v-for = 'item in items':item = 'item'></slot>
46.     </div>
47.     `,
48.     });
49.     app.mount("#app");           // 实例挂载
50. </script>
51. </body>
52. </html>

```

上述代码中,第 16~20 行使用子组件 child,并在其中通过带 v-slot:default 指令的 template 元素使用作用域插槽,将包含所有插槽 prop 的对象命名为 props(临时变量),并使用 props.item.xxx 的形式获取子组件提供的数据;第 22~26 行使用子组件 child,并在其中通过 v-slot:user 指令的 template 元素使用作用域插槽;第 31~48 行全局定义子组件 child,在模板选项中插入两个插槽(一个匿名、一个具名),绑定 item 属性,循环遍历 items 数组变量中的所有数据,其中 v-for='item in items'中的 item 是临时变量。而绑定在插槽 slot 元素上的 item 属性称为“插槽 prop”,在父级作用域中,可以使用带值的 v-slot 指令定义子组件提供的插槽 prop 的名称,方法如第 18 行和第 24 行中的 {{props.item.id}}。

5.3.4 动态插槽名

动态指令参数也可以用在 v-slot 上,定义动态插槽名。部分示例代码如下。

```

<my-child>
  <template v-slot:[dynamicSlotName]>
    ...
  </template>
</my-child>

```

动态插槽名需要使用方括号来包裹,dynamicSlotName 是一个变量,然后在 App 组件的 data 选项中定义动态插槽名,并赋初值。通过其他事件改变动态插槽名,达到动态渲染子组件。

【例 5-13】 动态插槽名实战。代码如下,页面效果如图 5-18 和图 5-19 所示。

```

1. <!-- vue-5-13.html -->
2. <!DOCTYPE html>
3. <html>
4.   <head>
5.     <meta charset = "UTF-8" />
6.     <title>动态插槽名的应用</title>
7.     <script src = "../js/vue.global.js" type = "text/javascript"></script>
8.   </head>
9.   <body>
10.    <div id = "app">
11.      <h3>插槽名动态更新</h3>
12.      <child>
13.        <template v-slot:[dynslot]>
14.          <h3>动态切换插槽:显示重要信息!</h3>
15.        </template>
16.      </child>
17.      <button @click = "changeSlot">更新插槽名称</button>
18.    </div>
19.    <script>
20.      const child = {
21.        template: `

```

```
22.     <div>
23.       <p><slot name = 'slot1'>插槽 1:欢迎使用动态插槽!</slot></p>
24.       <p><slot name = 'slot2'>插槽 2:欢迎使用动态插槽!</slot></p>
25.     </div>
26.   `,
27.   };
28.   const App = {
29.     data() {
30.       return {dynslot: "", flag: true,};
31.     },
32.     components: { child, },
33.     methods: {
34.       changeSlot() { // 先根据 flag 值,给动态参数赋值为指定插槽名
35.         this.dynslot = this.flag ? "slot1" : "slot2";
36.         this.flag = !this.flag; // 然后 flag 值取反,为下次切换做准备
37.         console.log(this.flag + " -- " + this.dynslot); // 控制台输出
38.       },
39.     },
40.   };
41.   const app = Vue.createApp(App); // 创建 Vue 实例
42.   app.mount("#app"); // 挂载
43. </script>
44. </body>
45. </html>
```

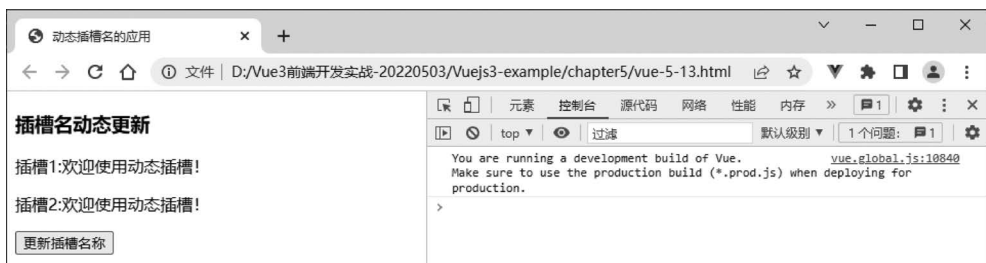


图 5-18 动态插槽名默认为空时的页面效果

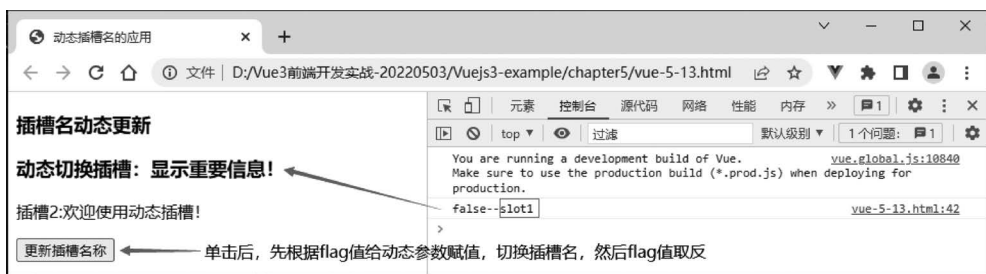


图 5-19 单击按钮后动态插槽名被重新赋值后的页面效果

当第1次单击“更新插槽名称”按钮时,flag 值为 true,将 slot1 赋给了 dynslot,此时插槽 1 的默认内容被父组件传递的内容所替换,而插槽 2 显示的还是默认信息,如图 5-19 所示。当第 2 次单击“更新插槽名称”按钮时,flag 值为 false,将 slot2 赋给了 dynslot,此时插槽 2 的默认内容被父组件传递的内容所替换,而插槽 1 显示的还是默认信息。继续单击按钮会继续进行动态赋值。

本章小结

本章主要介绍了组件基础、组件间通信和插槽。

在组件基础中,重点介绍了组件的命名规范和注册方法。Vue 组件中通常采用 camelCase(驼峰式)或 kebab-case(短横线分隔式)命名方式。组件可以全局注册,也可以局部注册。

在组件间通信中,主要讲解了父组件向子组件传值、子组件向父组件传值和父链与子组件索引等。在子组件中通过 props 使用父组件传递过来的数据,这是单向传递,只能从父组向子组件传递。子组件用 \$emit() 触发自定义事件,父组件通过 \$on() 或 v-on:eventname 侦听子组件的事件,父组件也可以使用 v-model 指令通过 input 事件传递数据。当然,也可以通过父链与子组件索引传递数据。

在插槽中,重点讲解了匿名插槽、具名插槽、作用域插槽及动态插槽名的应用场景和基本语法。

扫一扫



自测题

练习 5

1. 选择题

- (1) 下列组件命名符合驼峰式命名方式的是()。
 - A. MYCOMPONENET
 - B. mycomponent
 - C. MyComponent
 - D. myComponent
- (2) 在 Vue 3.0 中,父组件向子组件传递两个数据 data1 和 data2 时,子组件下正确的声明方式是()。
 - A. props:['data1','data1']
 - B. props:{'data1','data1'}
 - C. props:[data1:String,data1:Number]
 - D. props:(data1,data2)
- (3) 当父组件使用 v-model 指令侦听事件时,子组件中可以使用()发送数据。
 - A. this.\$emit('input',Data)
 - B. this.emit('input',Data)
 - C. this.\$emit('v-model',Data)
 - D. this.emit('v-model',Data)
- (4) 下列具名(slotName)插槽缩写方式中正确的是()。
 - A. v-slot: slotName
 - B. :slotName
 - C. #slotName
 - D. @slotName
- (5) 父组件通过元素的 slot 属性可以将数据信息传递至指定()属性的具名插槽中。
 - A. name
 - B. slot
 - C. v-slot
 - D. slot-scope
- (6) 下列动态插槽名(dynSlotName)定义中正确的是()。
 - A. <template v-slot:dynSlotName>
 - B. <template v-slot:"dynSlotName">
 - C. <template slot=[dynSlotName]>
 - D. <template v-slot:[dynSlotName]>

2. 填空题

- (1) 全局注册组件必须使用 _____ 指令来实现,局部注册组件必须在 App 实例的 _____ 选项中注册。
- (2) 在 HTML 模板中,myComponent 组件只能使用 _____ 命名方式,引用格式如 _____。

(3) 在单组件文件的 template 元素中,引用 PascalCasedComponent 组件,可能的格式有:

_____、_____、_____。

(4) 插槽通常分为_____、具名插槽、作用域插槽等。父组件可以使用 slot 属性将信息插入指定的具名插槽中。

(5) v-slot 指令一般添加在_____元素上。其他元素上可以使用_____属性来设置。

3. 简答题

(1) 简述组件中 data 与 props 属性在使用上的区别。

(2) 简述兄弟组件间通信常用的方法。

项目实战 5

1. 父子组件间通信实战——计算任意区间连续整数累加和

【实战要求】

- (1) 学会定义 Vue 组件,并学会全局注册组件。
- (2) 学会使用 CSS 定义<div>、<button>、<input>标记的样式。
- (3) 学会使用父子组件通信实现两个组件间相互通信,能够使用 props、\$emit() 传递数据。
- (4) 学会在 Vue 3.x 中使用 Options API 定义选项。
- (5) 学会在<body>标记中定义模板(<template id='templ'></template>)供组件引用(通过 ID 号引用,如 template:'#templ')。

【设计要求】

参照如图 5-20 所示页面,完成“计算任意区间连续整数累加和”项目设计。具体要求如下。



图 5-20 计算任意区间连续整数累加和初始界面

(1) 定义两个组件。组件名称分别为 sendData 和 receiveData,组件的内容如图 5-20 所示。

(2) 在 sendData 组件中,在两个文本框中输入起始数和终止数后,通过 props 下发给 receiveData 组件,receiveData 组件中数据同步渲染。单击“计算累加和并发送给父组件”按钮后,执行事件处理函数 sum(number1,number2),完成传递数据合法性检查、累加和计算和发送结果。当数据合法检查通过后,将计算结果显示在子组件中,同时发送数据给 sendData 组件,累加和会同步渲染在 receiveData 组件中,如图 5-21 所示。

扫一扫



视频讲解



图 5-21 输入数据合法情况下的页面效果

(3) 当父组件 sendData 中输入的起始数大于或等于终止数时,子组件 receiveData 会向父组件传递错误消息,并在父组件相关的<div>标记中显示红色的错误信息“父组件中起始数不能大于或等于终止数! 请重新输入数据!”,如图 5-22 所示。



图 5-22 输入数据不合法情况下的页面效果

(4) 页面样式要求如下。

- 父组件 #app 样式: 有边界(上下为 0,左右自动),宽度为 600px,高度为 370px,有边框(15px,实线, #CACACA),内容居中对齐,圆角边框(半径为 25px)。
- <button>和<input>样式: 高度为 24px,圆角边框(半径为 10px),有边框(1px,虚线, #0000EE)。

【实战步骤】

(1) 建立 HTML 文件。项目文件名为 vue-ex-5-1.html,引用 vue.global.js 文件。在<body>标记中插入 id 为 app 的<div>,并在其中引用组件<send-data></send-data>。

(2) 定义组件。按图 5-20~图 5-22 效果分别定义 sendData、receiveData 组件。其中,子组件 receiveData 发送计算结果数据和出错信息。

```
this.$emit("compute", this.result); // 数据合法,发送计算结果
this.$emit("error", "父组件中起始数不能大于或等于终止数!请重新输入数据!");
```

(3) 组件通信测试。在 sendData、receiveData 组件中分别输入相关内容后,单击“计算累加和并发送给父组件”按钮,查看运行结果,调试并完善代码。

2. 插槽综合实战——父子组件数据传递

【实战要求】

- (1) 学会引用 vue.global.js 文件,学会定义 Vue 组件,构建基本 HTML 文件。
- (2) 学会使用 CSS 定义<div>、<button>、等标记的样式。
- (3) 利用 props 属性实现父组件向子组件传值,子组件通过 \$emit()向父组件发送数据。
- (4) 学会定义匿名插槽、具名插槽和作用域插槽,利用作用域插槽向父组件提供数据。
- (5) 学会创建 Vue 实例和配置相关选项,会定义相关方法实现相关功能。

【设计要求】

参照如图 5-23 和图 5-24 所示的页面,完成项目设计。具体设计要求如下。



图 5-23 插槽综合应用初始页面



图 5-24 单击“修改数据”按钮后页面效果

- (1) 在父组件中定义使用子组件标记,其标记为<son1></son1>,并绑定 name、age、sex 属性值。侦听自定义事件 update-data,绑定事件处理函数 updateData()。定义两个 template 模板,插槽名分别为 top 和 bottom,第 2 个模板为作用域插槽,使用子组件中绑定的 courses

扫一扫



视频讲解

对象中的数据。

(2) 在子组件中插入 3 个插槽,名称分别为 top、default、bottom。

(3) 在匿名插槽中显示姓名、年龄、性别等信息,并实现“修改数据”按钮的功能。单击“修改数据”按钮后,执行 `updateData()` 函数,将姓名、年龄和性别封装在 `data1` 对象中,并发送给父组件。父组件中显示同步渲染,更新后页面效果如图 5-24 所示。作用域插槽中绑定的课程信息为“计算机组成原理,56 元”“Vue.js 前端框架技术与实战,69.8 元”“Web 前端开发技术实验与实践,45 元”。

【实战步骤】

(1) 建立 HTML 文件。项目文件名为 `vue-ex-5-2.html`,引用 `vue.global.js` 文件。在 `<body>` 标记中分别插入 `<div>`、`<template>` 和 `<script>` 标记,并在 `<div>` 中插入子组件 `son1`。

(2) 定义父组件。按照设计要求(1)完成父组件的设计。

(3) 定义子组件。要求使用 `app.component("son1", {})` 全局注册子组件 `son1`。按照设计要求(2)和设计要求(3)完成子组件的设计,同时通过父组件向子组件传值(`name`、`age`、`sex`),供子组件使用。

(4) 定义样式。

- 子组件 `son1` 中 `<div>` 的 `son1` 类样式: 内容居中显示,有边界(1px,虚线,#1B5764)。
- `<button>` 样式: 宽度为 200px,高度为 35px,有边界(1px,虚线,#CCCCCC),圆角边框(半径为 20px)。
- `<ul>` 样式: 列表样式类型为 none。
- 插槽中 `<div>` 的 `box` 类样式: 背景颜色为 #DAFBFC。

(5) 调试页面并运行。代码设计完成后,通过浏览器运行查看页面效果,调试并完善代码。