



## 第 3 章 Vue 事件、组件及生命周期

Vue 中有很多 Vue WebUI 组件库可供开发者使用,那么组件是如何开发出来的? 针对组件的事件处理又是如何描述的? 本章将对 Vue 基础知识进行讲解,内容包括 Vue 事件处理、Vue 组件、Vue 生命周期等。

### 本章要点

- 掌握 Vue 的事件监听操作
- 掌握 Vue 组件的定义和注册方法
- 掌握 Vue 组件直接传递数据的方法
- 掌握 Vue 生命周期钩子函数的使用方法

### 励志小贴士

人生总要经历起起伏伏,不要因为一两次的失败就郁郁寡欢。打磨自己的过程总是充满了艰难和迷茫,要相信:坚持的人,一定能找到属于自己的亮光。

## 3.1 Vue 事件

可以使用 `v-on` 指令(通常缩写为符号`@`)监听 DOM 事件,并在触发事件时执行一些 JavaScript。用法为“`v-on:事件名="方法"`”或使用快捷方式“`@事件名="方法"`”。之前的案例使用过`@click`、`@keyup.enter` 等,下面详细介绍这些内容。

### 3.1.1 事件监听

在 Vue 中,可以使用内置指令 `v-on` 监听 DOM 事件,下面通过例 3-1 进行演示。

**【例 3-1】** 事件监听。

(1) 创建文件夹 `chapter03`,然后在该目录下创建 `demo01.html` 文件,具体代码如下:

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>Title</title>
6  </head>
7  <body>
8    <div id="app">
9      <div>{{count}}</div>
10     <button type="button" @click="count+=1">+1</button>
11     <p>
12       <button type="button" @click="showDt">当前日期时间</button>
13       {{now}}
14     </p>
15   </div>
16   <script src="vue.js"></script>
17   <script>
18     const app = Vue.createApp(
19       {
20         data() {
21           return {
22             count: 0,
23             now: ''
24           }
25         },
26         methods: {
27           showDt: function () {
28             this.now = new Date()
29           }
30         }
27       }
30     )
31     app.mount('#app')
```

```
31     })
32     app.mount('#app')
33   </script>
34 </body>
35 </html>
```

(2) 在浏览器中打开 demo01.html 文件,运行结果如图 3-1 所示。单击按钮后,运行结果如图 3-2 所示。

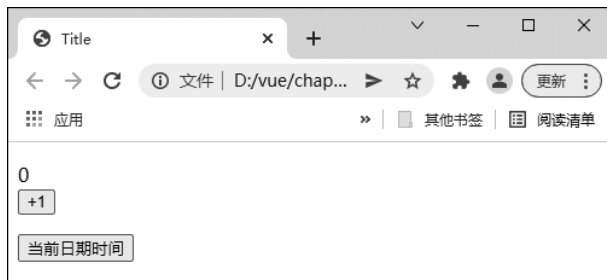


图 3-1 初始结果

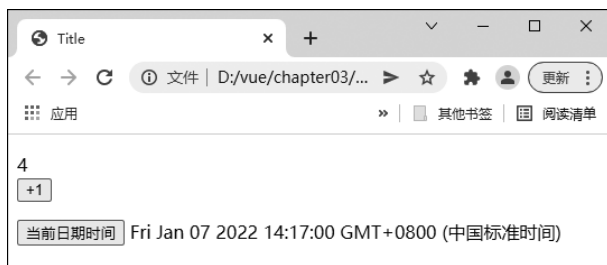


图 3-2 单击按钮后的运行结果

### 3.1.2 事件修饰符

事件修饰符是自定义事件行为,配合 v-on 指令使用,写在事件之后,用“.”符号连接,如 v-on:click.stop 表示阻止事件冒泡。

示例如下:

```
1   <!--阻止单击事件冒泡 -->
2   <a v-on:click.stop="doSth"></a>
3   <!--阻止事件默认行为 -->
4   <form v-on:submit.prevent="onSubmit"></form>
5   <!--修饰符串联 -->
6   <a v-on:click.stop.prevent="doSth"></a>
7   <!--只有修饰符 -->
8   <form v-on:submit.prevent></form>
9   <!--添加事件监听器时使用事件捕获模式 -->
```

```
10 <a v-on:click.capture="doSth"></a>
11 <!-- 只当事件在该元素本身触发时触发回调 -->
12 <div v-on:click.self="doSth"></div>
13 <!-- 事件只触发一次 -->
14 <a v-on:click.once="doSth"></a>
```

### 3.1.3 按键修饰符

在监听键盘事件时,经常需要检查常见的键值。为了方便开发,Vue 允许为 v-on 添加按键修饰符以监听按键,如 Enter、Space、Shift 和 Down 等。下面以 Enter 键为例进行演示。

**【例 3-2】** 按键修饰符的使用。

(1) 创建 chapter03/demo02.html 文件,具体代码如下:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>标题</title>
6 </head>
7 <body>
8   <div id="app">
9     输入后按 Enter 键则提交: <input type="text" v-on:keyup.enter="submit">
10  </div>
11 <script src="vue.js"></script>
12 <script>
13   const app = Vue.createApp(
14     {
15       methods: {
16         submit() {
17           console.log('表单提交')
18         }
19       }
20     })
21   app.mount('#app')
22 </script>
23 </body>
24 </html>
```

上述代码中,当按 Enter 键后,就会触发 submit() 事件处理方法。

(2) 在浏览器中打开 demo02.html,单击 input 输入框使其获得焦点,然后按 Enter 键,运行结果如图 3-3 所示。从图 3-3 中可以看出,控制台输出了“表单提交”,说明键盘事件绑定成功且执行。

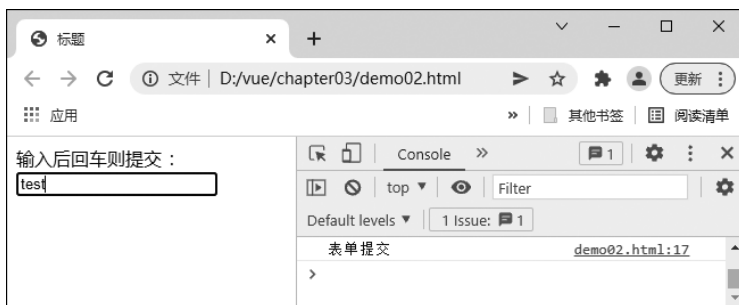


图 3-3 按 Enter 键触发事件

## 3.2 Vue 组件

Vue 可以进行组件化开发,组件是 Vue 的基本结构单元,在开发过程中使用起来非常方便灵活,只需要按照 Vue 规范定义组件,将组件渲染到页面即可。组件能实现复杂的页面结构,提高代码的可复用性。在开源社区,有很多 Vue WebUI 组件库可供开发者使用。例如,ElementUI 就是一套基于 Vue.js 的高质量 UI 组件库,可以用其快捷地开发前端界面。下面对 Vue 组件进行讲解。

### 3.2.1 什么是组件

在 Vue 中,组件是构成页面中独立结构的单元,能够减少代码的重复编写,提高开发效率,降低代码之间的耦合度,使项目更易维护和管理。组件主要以页面结构的形式存在,不同组件也具有基本的交互功能,可以根据业务逻辑实现复杂的项目功能。

下面通过一个案例演示组件的定义和使用。

**【例 3-3】** 组件的定义和使用。

(1) 创建 chapter03/demo03.html 文件,具体代码如下:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>创建并注册组件</title>
6 </head>
7 <body>
8 <div id="app">
9   <my-com></my-com>
10  <hr/>
11  <my-com></my-com>
12 </div>
13 <script src="vue.js"></script>
```

```
14 <script>
15   const app = Vue.createApp({})
16   app.component('myCom', {
17     template: '<button type="button"
18 @click="btnHandler">{{msg}}</button>', data() {
19     return {
20       msg: '自定义组件'
21     }
22   },
23   methods: {
24     btnHandler() {
25       alert('haha~~');
26     }
27   }
28 });
29 app.mount('#app')
30 </script>
31 </body>
32 </html>
```

在上述代码中,第 16 行的 `app.component()` 表示注册组件的 API,参数 `myCom` 为组件名称,该名称与页面中的 `<my-comp>` 标签名对应;第 17 行的 `template` 表示组件的模板;第 18~22 行表示组件中的数据,它必须是一个函数,并通过返回值返回初始数据;第 23~28 行表示组件中的方法。

(2) 在浏览器中打开 `demo03.html`,运行结果如图 3-4 所示。



图 3-4 自定义组件的运行结果

如图 3-4 所示,一共有两个 `my-comp` 组件,单击某一个组件时,会显示一个弹框。

通过例 3-3 可以看出,利用 Vue 的组件功能可以非常方便地复用页面代码,实现一次定义、多次使用的效果。

### 3.2.2 局部注册组件

前面学习的 `app.component()` 方法用于全局注册组件,除了全局注册组件外,还可以

局部注册组件,即通过 Vue 实例的 component 属性实现。下面通过例 3-4 进行演示。

**【例 3-4】** 局部注册组件。

(1) 创建 chapter03/demo04.html 文件,具体代码如下:

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>局部组件</title>
6  </head>
7  <body>
8    <div id="app">
9      <my-com></my-com>
10     <hr/>
11     <my-com2></my-com2>
12   </div>
13   <template id="tem">
14     <div>
15       <p>局部组件 2 -- {{count}}</p>
16       <button type="button" @click="btnHandler">单击</button>
17     </div>
18   </template>
19   <script src="vue.js"></script>
20   <script>
21     const app = Vue.createApp({})
22     //定义一个普通的 JavaScript 对象
23     const Com = {
24       template: '<h3>局部组件- <input v-model="msg">- {{msg}}</h3>',
25       data() {
26         return {msg: 'hello'}
27       }
28     }
29     const Com2 = {
30       template: '#tem',
31       data() {
32         return {
33           count: 0
34         }
35       },
36       methods: {
37         btnHandler: function () {
38           this.count++;
39         }
40       }
41     }
```

```
42     app.component('myCom', Com)
43     .component('myCom2', Com2)
44     app.mount('#app')</script>
45 </body>
46 </html>
```

在上述代码中,第 42、43 行的 component 表示组件配置选项,注册组件时,只需要在 component 内部定义组件即可。

(2) 在浏览器中打开 demo04.html,运行结果如图 3-5 所示。

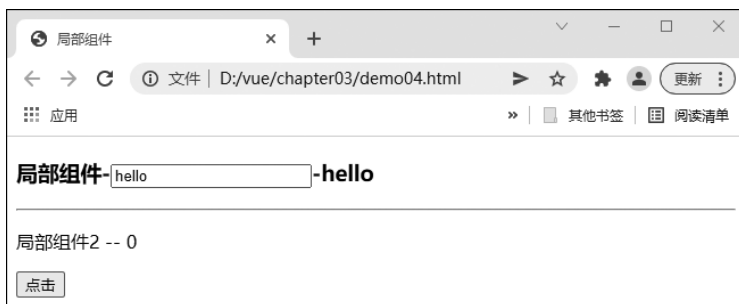


图 3-5 components 注册组件的运行结果

在上述代码中,可以看到 template 模板是使用字符串保存的,这种方式不仅容易出错,也不适合编写复杂的页面结构。实际上,模板代码是可以写在 HTML 结构中的,Vue 提供了<template>标签以定义结构的模板,可以在该标签中书写 HTML 代码,然后通过 id 值绑定到组件内的 template 属性上,例如代码第 14~19 行定义了模板 HTML 代码,第 32 行将该模板<template>绑定到了组件上。

### 3.2.3 组件之间的数据传递

在 Vue 中,组件实例具有局部作用域,组件之间的数据传递需要借助一些工具(如 props 属性)以实现从父组件向子组件传递数据信息。从父组件向子组件传递数据信息是从外部向内部传递,从子组件向父组件传递数据信息是从内部向外部传递。

在 Vue 中,数据传递主要通过 props 属性和 \$emit 方式实现,下面分别进行讲解。

#### 1. props 传值

props 即道具。组件实例的作用域是孤立的,这意味着不能且不应在子组件的模板内直接引用父组件的数据。通常可以使用 props 把数据传给子组件。下面具体演示 props 属性的使用。

**【例 3-5】** props 属性的使用。

(1) 创建 chapter03/demo05.html 文件,具体代码如下:

```
1     <!DOCTYPE html>
2     <html lang="en">
```

```
3   <head>
4     <meta charset="UTF-8">
5     <title>props</title>
6   </head>
7   <body>
8     <div id="app">
9       <p>父组件 title: {{title}} ---num:
10        <input v-model="num"></p>
11       <hr/>
12       <son1 v-bind:son1-title="title" v-bind:son1-num="num"></son1>
13       <hr/>
14       <son2:title="title":num="num":obj="user":arr="msg"></son2>
15     </div>
16     <!--子组件利用 props 声明属性,父组件加载子组件时为属性赋值 -->
17     <!--子组件模板 -->
18     <template id="son2">
19       <div>
20         <p>我是子组件 2,以下演示来自父组件数据 :</p>
21         <p>{{title}}--{{num * 2}} --{{obj}} --{{obj.name}} --{{arr}}</p>
22       </div>
23     </template>
24     <script src="vue.js"></script>
25     <script>
26       const app = Vue.createApp({
27         data() {
28           return {
29             title: '使用 props 父组件向子组件传参',
30             num: 3,
31             msg: ['props', 'emit', 'bind', 10],
32             user: {
33               id: 1,
34               name: 'props'
35             }
36           }
37         })
38       //定义子组件
39       const son1 = {
40         template: '<div>来自父组件 title 与 num: {{son1Title}}:
41 {{son1Num}}</div>',
42         props: ['son1Title', 'son1Num']
43       };
44       const son2 = {
```

```
45     template: '#son2',
46     props: {
47         title: String,
48         num: {
49             type: Number,
50             required: true,
51             default: 11,
52             validator: function (value) {
53                 return value > 0;
54             }
55         },
56         obj: {
57             type: Object,
58             default: function () {
59                 return {
60                     id: 1, name: 'admin'
61                 }
62             }
63         },
64         arr: {
65             type: Array,
66             default: function () {
67                 return ['apple', 'banana']
68             }
69         }
70     }
71 };
72 app.component('son1', son1)
73   .component('son2', son2)
74 app.mount("#app")
75 </script>
76 </body>
77 </html>
```

上述代码声明了两个子组件 son1 与 son2, 其中, son1 子组件为了使用父组件的数据, 必须先定义 props 属性, 即第 42 行“props: ['son1Title', 'son1Num']”, 此处仅仅是声明两个属性, 没有对属性使用任何约束, 在第 12 行中, 使用 v-bind 将父组件数据通过已定义的 props 属性传递给了子组件。需要注意的是, 在子组件中定义 props 时, 使用了 CamelCase 命名法。由于 HTML 不区分大小写, 因此当 camelCase 的 props 用于特性时, 需要将其转为 kebab-case(连字符隔开)。例如, 在 props 中定义的 myName 在用作特性时, 需要将其转换为 my-name。在父组件中使用子组件时, 可以通过以下语法将数据传递给子组件:

```
1    <子组件 v-bind:子组件属性=父组件数据属性></子组件>
```

可以为组件的 props 指定验证要求。如果有一个要求没有被满足,则 Vue 会在浏览器控制台发出警告。为了定制 props 的验证方式,可以为 props 中的值提供一个带有验证要求的对象,而不是一个字符串数组,如上述代码中定义的子组件 son2,son2 中定义的 props 属性为第 33~57 行;其中,属性 num 通过 type 定义了类型,通过 required:true 要求必须为该属性赋值,default 定义了默认值,validator 定义了验证要求;属性 obj 和 arr 各自定义了类型和默认值。

需要注意的是,当为对象和数组定义默认值时,必须使用函数返回,如上述代码的第 58~63 行和第 66~68 行所示。

(2) 在浏览器中打开 demo05.html,运行结果如图 3-6 所示。

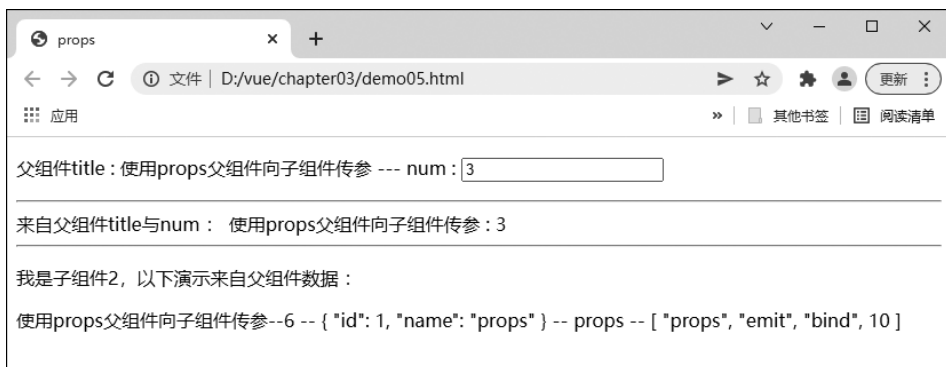


图 3-6 props 值的运行结果(1)

在图 3-6 所示的页面中,子组件显示标题为“使用 props 父组件向子组件传参”以及数字 3,说明父组件信息已经传递到子组件。当更新父组件 num 的值时,子组件中的数据也会随之发生改变,如图 3-7 所示。

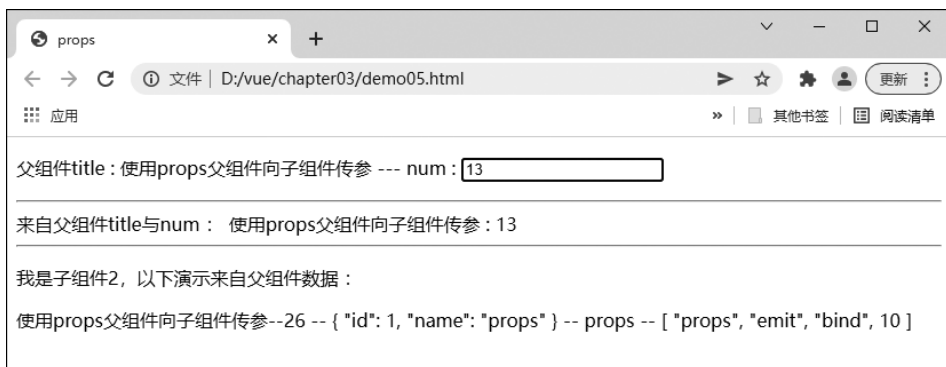


图 3-7 props 值的运行结果(2)

需要注意的是,props 是以从上到下的单向数据流传递的,且父级组件的 props 更新会向下流动到子组件中,但是反过来则不行,这是为了防止子组件无意中修改父组件的状态。

props 的 type 可以是下列原生构造函数中的一个：

- String
- Number
- Boolean
- Array
- Object
- Date
- Function
- Symbol

此外, type 还可以是一个自定义的构造函数。

## 2. \$emit 传值

\$emit 能够将子组件中的值传递到父组件中。\$emit 可以触发父组件中定义的事件,子组件的数据信息通过传递参数的方式完成。下面通过例 3-6 进行代码演示。

**【例 3-6】** \$emit 传值的使用。

(1) 创建 chapter03/dem06.html 文件,具体代码如下:

```
1    <!DOCTYPE html>
2    <html lang="en">
3    <head>
4        <meta charset="UTF-8">
5        <title>props</title>
6        <script src="vue.js"></script>
7    </head>
8    <body>
9    <div id="app">
10        <p>我是父组件--来自子组件的数据为: <br/>{{fromSon}}</p>
11        <hr/>
12        <son @son-msg="getDataFromSon"></son>
13    </div>
14    <template id="son">
15        <div>
16            <p>我是子组件</p>
17            <input type="text" v-model="msg"/>--{{msg}}
18            <button type="button" @click="toParent">将数据传递到父组件
19        </button>
20        </div>
21    </template>
22    <script>
23        const app = Vue.createApp({
24            data() {
25                return {
```

```
25         fromSon: ''
26     }
27 },
28     methods: {
29         getDataFromSon: function (sonData) {
30             this.fromSon = sonData;
31         }
32     }
33 })
34 const son = {
35     template: '#son',
36     data() {
37         return {
38             msg: '子组件字符串'
39         }
40     },
41     methods: {
42         toParent() {
43             this.$emit('son-msg', this.msg);
44         }
45     }
46 }
47 app.component('son', son)
48 .mount("#app")
49 </script>
50 </body>
51 </html>
```

上述代码的第 12 行,即在父组件中调用子组件时,绑定了一个自定义事件和对应的处理函数 `@son-msg="getDataFromSon"`;在第 43 行,子组件把要发送的数据通过触发自定义事件传递给父组件 `this.$emit('son-msg',this.msg)`;其中, `$emit()` 的意思是把事件沿着作用域链向上派送。

(2) 在浏览器中打开 `demo06.html` 文件,运行结果如图 3-8 所示。单击【将数据传递到父组件】按钮,运行结果如图 3-9 所示。

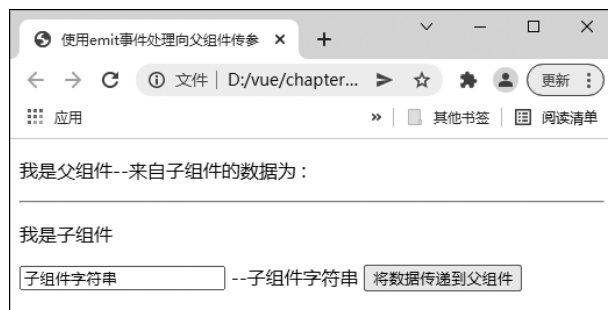


图 3-8 初始页面



图 3-9 传值成功

如图 3-8 所示,单击【将数据传递到父组件】按钮后,页面中显示了“子组件字符串”,说明成功完成了子组件向父组件的传值。

### 3.2.4 组件切换

Vue 中的页面结构是由组件构成的,不同组件可以表示不同页面,适合进行单页应用开发。下面通过例 3-7 演示登录组件和注册组件的切换。

**【例 3-7】** 组件切换。

(1) 创建 chapter03/demo07.html 文件,具体代码如下:

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>组件切换</title>
6      <script src="vue.js"></script>
7  </head>
8  <body>
9  <div id="app">
10     <a href="" @click.prevent="flag=true">登录</a>
11     <a href="" @click.prevent="flag=false">注册</a>
12     <login v-if="flag"></login>
13     <register v-else="flag"></register>
14 </div>
15 <script>
16     const app = Vue.createApp({
17         data() {
18             return {
19                 flag: true
20             }
21         }
22     })

```

```
23     app.component('login', {
24       template: '<h3>登录账号</h3>'
25     }).component('register', {
26       template: '<h3>注册账号</h3>'
27     }).mount("#app")</script>
28 </body>
29 </html>
```

上述代码中,第 12 行的 login 表示登录组件,第 13 行的 register 表示注册组件;第 12 行的 v-if 指令值为 true,表示加载当前组件,否则移除当前组件;第 10、11 行的.prevent 事件修饰符用于阻止<a>标签的超链接默认行为。

(2) 在浏览器中打开 demo07.html 文件,运行结果如图 3-10 所示。在页面中单击“注册”链接后,运行结果如图 3-11 所示。

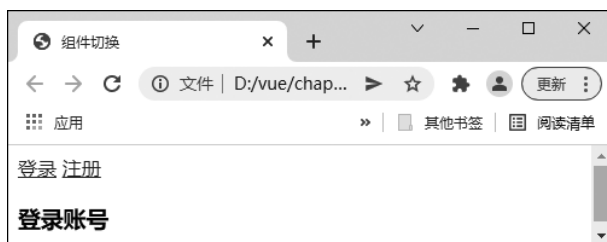


图 3-10 初始页面

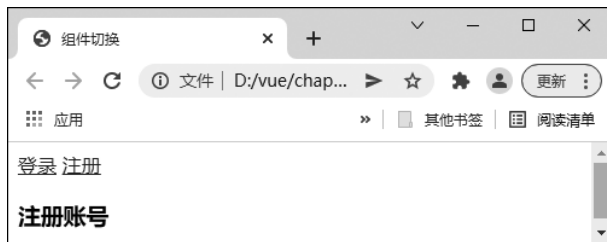


图 3-11 注册页面

从例 3-7 可以看出,组件的切换是通过 v-if 控制的。除了这种方式外,还可以通过组件的 is 属性实现,即使用 is 属性匹配组件的名称,下面通过例 3-8 进行演示。

**【例 3-8】** is 属性的使用。

(1) 创建 chapter03/demo08.html 文件,具体代码如下:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>组件切换</title>
6   <script src="vue.js"></script>
```

```
7     </head>
8     <body>
9     <div id="app">
10         <a href="" @click.prevent="comName='login'">登录</a>
11         <a href="" @click.prevent="comName='register'">注册</a>
12         <component v-bind:is="comName"></component>
13     </div>
14     <script>
15         const app = Vue.createApp({
16             data() {
17                 return {
18                     comName: 'login'
19                 }
20             }
21         })
22         app.component('login', {
23             template: '<h3>登录组件</h3>'
24         }).component('register', {
25             template: '<h3>注册组件</h3>'
26         }).mount("#app")
27     </script>
28 </body>
29 </html>
```

在上述代码中,第 12 行的 `is` 属性值绑定了 `data` 中的 `comName`;第 10、11 行的 `<a>` 标签用来修改 `comName` 的值,从而切换对应的组件。

(2) 在浏览器中打开 `demo08.html` 文件,运行结果与图 3-10 所示相同。

### 3.3 Vue 生命周期

Vue 实例为生命周期提供了回调函数,用来在特定的情况下触发,贯穿了 Vue 实例化的整个过程,这给用户在不同阶段添加自己的代码提供了机会。每个 Vue 实例在被创建时都要经过一系列的初始化过程,如初始数据监听、编译模板、将实例挂载到 DOM、在数据变化时更新 DOM 等。

Vue 的生命周期分为 4 个阶段,涉及 7 个函数。

- **create 创建**: `setup()`。
- **mount 挂载**(把视图和模型关联起来): `onBeforeMount()`, `onMounted()`。
- **update 更新**(模型的更新对视图造成何种影响): `onBeforeUpdate()`, `onUpdated()`。
- **unMount 销毁**(视图与模型失去联系): `onBeforeUnmount()`, `onUnmounted()`。

#### 3.3.1 钩子函数

钩子函数用来描述 Vue 实例从创建到销毁的整个生命周期,具体如表 3-1 所示。

表 3-1 生命周期钩子函数

| 钩 子             | 说 明                                                       |
|-----------------|-----------------------------------------------------------|
| setup           | 开始创建组件之前,在 beforeCreate 和 created 之前执行,创建的是 data 和 method |
| onBeforeMount   | 组件挂载到节点上之前执行的函数                                           |
| onMounted       | 组件挂载完成后执行的函数                                              |
| onBeforeUpdate  | 组件更新之前执行的函数                                               |
| onUpdated       | 组件更新完成之后执行的函数                                             |
| onBeforeUnmount | 组件卸载之前执行的函数                                               |
| onUnmounted     | 组件卸载完成后执行的函数                                              |

下面对这些钩子函数分别进行讲解。

### 3.3.2 实例创建

setup(): beforeCreate 和 created 与 setup 几乎是同时进行的,所以可以把写在 beforeCreate 和 created 这两个周期的代码直接写在 setup 中。

**【例 3-9】** 实例创建。

(1) 创建 chapter03/demo09.html 文件,具体代码如下:

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>钩子函数</title>
6      <script src="vue.js"></script>
7  </head>
8  <body>
9      <div id="app">
10         <input v-model.lazy="msg"/>
11         <button type="button" @click="btnHandler">{{msg}}</button>
12     </div>
13     <script>
14         const app = Vue.createApp({
15             data() {
16                 return {
17                     msg: 'helloworld',
18                 }
19             },
20             methods: {
21                 btnHandler: function () {
22                     console.log('button click');
23                 }
24             },

```

```
25     setup() {
26         console.log('setup() ----- ');
27         console.log(this.$el);      //undefined
28         console.log(this.$data);    //undefined
29         console.log(this.msg);      //undefined
30         alert('setup');
31     },
32 })
33 app.mount("#app")
34 </script>
35 </body>
36 </html>
```

(2) 在浏览器中打开 demo09.html 文件,运行结果如图 3-12 所示。

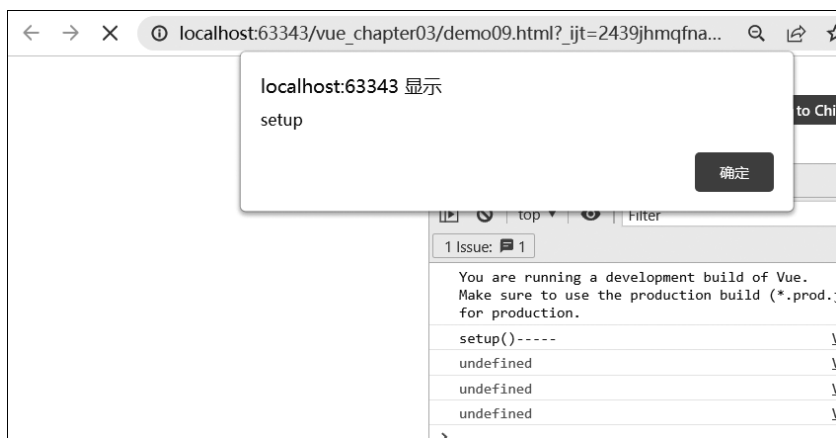


图 3-12 setup 的运行结果

如图 3-12 所示,setup 钩子函数输出 msg 时为 undefined,这是因为此时数据还没有被监听,同时页面上没有挂载对象。

### 3.3.3 页面挂载

onBeforeMount()表示模板已经在内存中编辑完成了,但是尚未把模板渲染到页面中。

onMounted()在这时挂载完毕,此时 DOM 节点已被渲染到文档内,一些需要 DOM 的操作在此时才能正常进行(常在此方法中进行 ajax 请求数据,渲染到 DOM 节点)。

**【例 3-10】** 页面挂载。

(1) 创建 chapter03/demo10.html 文件,具体代码如下:

```
1 <!DOCTYPE html>
2 <html lang="en">
```

```
3   <head>
4     <meta charset="UTF-8">
5     <title>钩子函数</title>
6     <script src="vue.js"></script>
7   </head>
8   <body>
9     <div id="app">
10      <input v-model.lazy="msg"/>
11      <button type="button" @click="btnHandler">{{msg}}</button>
12    </div>
13    <script>
14      const {onMounted, onBeforeMount, reactive,toRefs} = Vue
15      const app = Vue.createApp({
16        setup() {
17          const data = reactive({
18            msg: 'helloworld',
19          })
20          const methods = {
21            btnHandler: function () {
22              console.log('button click');
23            },
24          }
25          onBeforeMount(() => {
26            console.log('beforeMount()----');
27            let btn = document.querySelector('button')
28            console.log(btn)
29          })
30          onMounted(() => {
31            console.log('mounted()----');
32            let btn = document.querySelector('button')
33            console.log(btn)      //此时可以打印出 button 的值
34          })
35          return {
36            ...toRefs(data),
37            ...methods
38          }
39        }
40      })
41      app.mount("#app")
42    </script>
43  </body>
44  </html>
```

(2) 在浏览器中打开 demo10.html 文件,运行结果如图 3-13 所示。

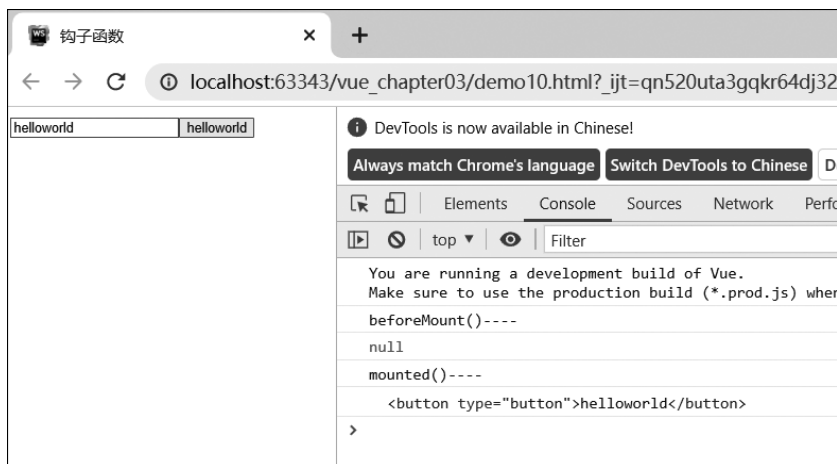


图 3-13 onBeforeMount 与 onMounted 的运行结果

从图 3-13 可以看出,在挂载之前,数据并没有被关联到对象上,所以页面无法展示页面数据;在挂载之后就获得了 msg 数据,并通过插值语法展示到页面中。

### 3.3.4 数据更新

onBeforeUpdate(): 当执行 beforeUpdate 时,页面中显示的数据还是旧的,此时 data 数据是最新的,页面尚未和最新的数据保持同步。

onUpdated(): 页面和 data 数据已经保持同步,都是最新的。

**【例 3-11】** 数据更新。

(1) 创建 chapter03/demo11.html 文件,具体代码如下:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>钩子函数</title>
6   <script src="vue.js"></script>
7 </head>
8 <body>
9   <div id="app">
10     <div v-if="isShow" ref="test">test</div>
11     <button @click="isShow=!isShow">更新</button>
12   </div>
13   <script>
14     const {onBeforeUpdate, onUpdated, ref} = Vue
15     const app = Vue.createApp({
16       setup() {
17         const test = ref()
```