第 3 章

Vue 基础语法

前端框架很多，无论选择哪个框架，对于前端处理的事情依旧是模板渲染、处理用户数据交互、事件的绑定等，只不过每个框架的理念和写法不同而已。Vue 通过声明一个唯一的页面实例 new Vue({...})来标记当前页面的 HTML 结构、数据展示及相关事件的绑定。

3.1 模板语法

Vue.js 使用了基于 HTML 的模板语法，允许开发者声明式地将 DOM 与底层 Vue 实例的数据进行绑定。所有 Vue.js 的模板都是合法的 HTML，所以能被遵循规范的浏览器和 HTML 解析器解析。

在底层的实现上，Vue 将模板编译成虚拟 DOM 渲染函数。结合响应系统，Vue 能够智能地计算出最少需要重新渲染多少组件，并把 DOM 操作次数减到最少。

简单地说：Vue 模板语法实现前端渲染，前端渲染即是把数据填充到 html 标签中。数据(来自服务器)＋模板(html 标签)＝前端渲染(产物是静态 html 内容)。

3.1.1 插值

插值就是将数据插入 html 文档中，包含文本、html 元素、元素属性等。

1. 文本插值

文本插值中用得最多的就是用双大括号的形式插入文本，如例 3-1 所示。

【例 3-1】 文本插值方式

```
//第 3 章/文本插值.html
<!DOCTYPE html>
<html>
<head lang="en">
<meta charset="UTF-8">
<title></title>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
```

```
<body>
<div id="app">
    <!-- 与实例中【data】的属性绑定在一起,并且数据实现同步 -->
    <h1>{{message}}</h1>
</div></body>
<script>
    // Vue 所做的工作就是把数据填充到页面的标签中
    var vm = new Vue({
        el: "#app",
        // data 模型数据,其值是一个对象
        data: {
            message: "I LOVE YOU"
        }
    })
</script>
</body>
</html>
```

在上面代码中,data 的值更新之后不需要操作 html,页面会自动更新数据。

也可以只绑定数据一次,以后在更新 data 的属性时不需要再更新页面数据,如例 3-2 所示。

【例 3-2】 使用 v-once 指令插值

```
//第 3 章/使用 v-once 指令插值.Html
<!DOCTYPE html>
<html>
<head lang="en">
<meta charset="UTF-8">
<title></title>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<div id="app">
<!-- v-once 只编译一次。显示内容之后不再具有响应功能 -->
<!-- v-once 的应用场景,如果显示信息后不需要再修改,可以使用 v-once 指令,这样可以提高
性能,因为 Vue 就不需要去监听它的变化了 -->
    <h1 v-once>{{message}}</h1>
</div></body>
<script>
    // Vue 所做的工作就是把数据填充到页面的标签里
    var vm = new Vue({
        el: "#app",
        // data 模型数据,其值是一个对象
        data: {
```

```
            message: "I LOVE YOU"
        }
    })
</script>
</body>
</html>
```

在上面代码中页面只会呈现 I LOVE YOU,当改变 data 中的 message 属性值时,页面将不再刷新。

2. HTML 插值

双大括号会将数据解释为普通文本,而非 HTML 代码。为了输出真正的 HTML,需要使用 v-html,如例 3-3 所示。

【例 3-3】 使用 v-html 指令将数据解释成 HTML 代码

```
//第 3 章/v - html 指令.html
<!DOCTYPE html>
<html>
<head lang = "en">
<meta charset = "UTF - 8">
<title></title>
<script src = "https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<div id = "app">
<!-- 内容按普通 HTML 插入,不会作为 Vue 模板进行编译,在网站上动态渲染任意 HTML 是非常危险的,因为容易导致 XSS 攻击 -->
    <h1 v - html = "msg"></h1>
</div>
</body>
<script>
    varvm = new Vue({
        el: "#app",
        data: {
            msg: "<span style = 'color:blue'>BLUE</span>"/* 可以使用 v - html 标签展示 HTML 代码。*/
        }
    })
</script>
</html>
```

上面代码将 msg 属性值作为 html 元素插入 h1 标签的子节点中。

3. 属性插值

Mustache 语法不能作用在 HTML attribute 上,遇到这种情况应该使用 v-bind:指令。

在开发的时候，有时属性不是固定不变的，有可能根据我们的一些数据动态地决定，例如图片标签(<img>)的 src 属性，我们可能从后端请求了一个包含图片地址的数组，需要将地址动态地绑定到 src 上，这时就不能简单地将 src 固定为某个地址。还有一个例子就是 a 标签的 href 属性。这时可以使用 v-bind:指令。其中，v-bind:可以缩写成冒号:，如例 3-4 所示。

【例 3-4】 使用 v-bind:指令绑定属性

```
//第 3 章/使用 v-bind 指令.html
<!DOCTYPE html>
<html xmlns:v-bind="http://www.w3.org/1999/xhtml">
<head lang="en">
<meta charset="UTF-8">
<title></title>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<div id="app">
    <img v-bind:src="imgUrl" alt=""/>
    <a :href="searchUrl">百度一下</a>
</div>
</body>
<script>
    var vm = new Vue({
        el: "#app",
        data: {
            imgUrl:'images/1.jpg',
            searchUrl:'http://www.baidu.com'
            }
    })
</script>
</html>
```

服务器请求过来的数据，我们一般都会在 data 那里中转一下，中转过后再把需要的变量绑定到对应的属性上面。

v-bind 除了可以在开发中用在有特殊意义的属性外(src、href 等)，也可以绑定其他一些属性，如与 class 和 style 绑定，如例 3-5 和 3-6 所示。

【例 3-5】 v-bind 动态绑定属性 class

```
//第 3 章/v-bind 动态绑定属性.html
<!DOCTYPE html>
<html lang="en" xmlns:v-bind="http://www.w3.org/1999/xhtml">
<head>
<meta charset="UTF-8">
<title>v-bind 动态绑定属性 class</title>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

```
</head>
<body>
<div id="app">
<p:class="{fontCol:isName,setBack:!isAge}" class="weight">{{name}}</p>
<i :class="addClass">{{name}}真好看!</i>
</div>
<script>
    var vm = new Vue({
        el:"#app",
    // 条件比较少
    data:{
        isName:true,
        isAge:false,
        name:"功守道"
    },
    /*当v-bind:class的表达式过长或者逻辑复杂时(一般当条件多于两个的时候),可以考虑采用计算属性,返回一个对象*/
    computed:{
        addClass:function(){
            return{
                checked:this.isName&&!this.isAge
            }
        }
    }
})
</script>
</body>
</html>
```

既然是一个对象,那么该对象内的属性就可能不唯一,但只要有一项为真,对应的类名就会存在。通过 v-bind 更新的类名和元素本身存在的类名不冲突,可以优雅地共存,如图 3-1 所示。

图 3-1 v-bind 动态绑定属性 class

【例 3-6】 v-bind 动态绑定属性 style

```
//第 3 章/v - bind 动态绑定属性 style.html
<!DOCTYPE html>
<html lang = "en" xmlns:v - bind = "http://www.w3.org/1999/xhtml">
<head>
<meta charset = "UTF - 8">
<title> v - bind 动态绑定属性 style </title>
<script src = "https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<div id = "app">
<div :style = "{'color': color,'fontSize':fontSize + 'px'}">修饰文本</div>
</div>
<script>
    var vm = new Vue({
        el: "#app",
        data: {
            color: 'red',
            fontSize: 24
        }
    })
</script>
</body>
</html>
```

在大多情况下,标签中直接写一长串属性的样式不便于阅读和维护,所以一般写在 dada 或 computed 里,代码如下:

```
<div id = "app">
<div :style = "styles">修饰文本</div>
</div>
<script>
Var vm = new Vue({
        el: "#app",
        data: {
           styles:{
            color: 'red',
            fontSize: 14 + 'px'
           }
        }
    })
</script>
```

应用多个样式对象时，可以使用数组语法，如例 3-7 所示。

【例 3-7】 v-bind 绑定多个 style 属性

```
//第 3 章/v - bind 绑定多个 style 属性.html
<!DOCTYPE html>
<html lang = "en" xmlns:v - bind = "http://www.w3.org/1999/xhtml">
<head>
<meta charset = "UTF - 8">
<title> v - bind 动态绑定属性 style </title>
<script src = "https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<div id = "app">
    <div :style = "[styleA,styleB]">文本</div>
</div>
<script>
    var vm = new Vue({
        el: "#app",
        data: {
            styleA:{
                color: 'red',
                fontSize: 24 + 'px'
            },
            styleB: {
                width: 100 + 'px',
                border: 1 + 'px ' + 'black ' + 'solid'
            }
        }
    })
</script>
</body>
</html>
```

4. 插值中使用 JavaScript 表达式

迄今为止，在模板中，我们一直都只绑定简单的 property 键值。但实际上，对于所有的数据绑定，Vue.js 都提供了完全的 JavaScript 表达式支持，如例 3-8 所示。部分表达式格式代码如下：

```
{{ number + 1 }}
{{ ok ? 'YES' : 'NO' }}
{{ message.split('').reverse().join('') }}
<div v - bind:id = "'list - ' + id"></div>
```

这些表达式会在所属 Vue 实例的数据作用域下作为 JavaScript 被解析。有个限制需要注意，每个绑定只能包含单个表达式，所以下面的表达式不会生效，代码如下：

```
<!-- 这是语句,不是表达式 -->
{{ var a = 1 }}
<!-- 流控制不会生效,请使用三元表达式 -->
{{ if (ok) { return message } }}
```

【例 3-8】 使用 JavaScript 表达式

```
//第 3 章/使用 JavaScript 表达式.html
<!DOCTYPE html>
<html lang="en" xmlns:v-bind="http://www.w3.org/1999/xhtml">
<head>
<meta charset="UTF-8">
<title></title>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<div id="app">
    <p>{{ number + 1 }}</p>
    <hr>
    <p>{{msg + '～～～～～'}}</p>
    <hr>
    <p>{{flag ? '条件为真' : '条件为假'}}</p>
    <hr>
</div>
<script>
    var vm = new Vue({
        el:'#app',
        data:{
            msg:'Hello beixi!',
            flag:true,
            number:2
        }
    })
</script>
</body>
</html>
```

3.1.2 指令

指令在上面已经使用过了，例如 v-bind:和 v-html，指令就是指这些带有 v-前缀的特殊属性。指令属性的值预期是单一 JavaScript 表达式(除了 v-for)。指令的职责就是当其表达式的值改变时相应地将某些行为应用到 DOM 上。

1. 参数

有一些指令能够接收一个“参数”，在指令名称之后以冒号表示。如 v-bind 指令可以用于响应式地更新 HTML attribute，代码如下：

```
<a v-bind:href = "url">...</a>
```

在这里 href 是参数，告知 v-bind 指令将该元素的 href attribute 与表达式 url 的值绑定。

2. 动态参数

从版本 2.6.0 开始，可以用方括号将 JavaScript 表达式作为一个指令的参数括起来，响应式使得 Vue 更加灵活多变，其动态参数也有其含义，代码如下：

```
<a v-bind:[attributeName] = 'url'>...</a>
```

这里的 attributeName 会被作为一个 JavaScript 表达式进行动态求值，求得的值将作为最终的参数使用。例如，如果 Vue 实例有一个 data 属性 attributeName，其值为'href'，那么这个绑定将等价于 v-bind:href。

同样地，可以使用动态参数作为一个动态的事件名绑定并处理函数，格式代码如下：

```
<a v-on:[eventName] = 'doSomething'>...</a>
```

同样地，当 eventName 的值为'focus'时，v-on:[eventName]等价于 v-on:focus。

当然动态参数的值也有约束，动态参数预期会求出一个字符串，异常情况下此值为 null。这个特殊的 null 值可以被显性地移出绑定。任何其他非字符串类型的值都会触发一个警告。

3. 修饰符

修饰符(Modifiers)是以半角句号“.”指明的特殊后缀，用于指出一个指令应该以特殊方式绑定。例如，.prevent 修饰符告诉 v-on 指令对于触发的事件调用 event.preventDefault()，代码如下：

```
<form v-on:submit.prevent = "onSubmit">...</form>
```

在后面章节对 v-on 和 v-for 等功能的探索中，将会看到修饰符的其他例子。

3.1.3 过滤器

过滤器是对即将显示的数据做进一步的筛选和处理，然后进行显示，值得注意的是过滤器并没有改变原来的数据，只是在原数据的基础上产生新的数据。

过滤器分全局过滤器和局部过滤器，全局过滤器在项目中使用频率非常高。

1. 定义过滤器

- 全局过滤器

```
Vue.filter('过滤器名称', function (value1[,value2,...] ) {
//逻辑代码
})
```

- 局部过滤器

```
new Vue({
        filters: {
    '过滤器名称': function (value1[,value2,...] ) {
       // 逻辑代码
           }
     }
})
```

2. 过滤器使用的地方

Vue.js 允许自定义过滤器，可被用于一些常见的文本格式化。过滤器可以用在两个地方：双花括号插值和 v-bind 表达式（后者从 2.1.0＋版开始支持）。过滤器应该被添加在 JavaScript 表达式的尾部，由“管道”符号指示，格式代码如下：

```
<!-- 在双花括号中 -->
<div>{{数据属性名称 | 过滤器名称}}</div>
<div>{{数据属性名称 | 过滤器名称(参数值)}}</div>
<!-- 在 v-bind 中 -->
<div v-bind:id="数据属性名称 | 过滤器名称"></div>
<div v-bind:id="数据属性名称 | 过滤器名称(参数值)"></div>
```

3. 实例

局部过滤器在实例中的使用如例 3-9 所示。

【例 3-9】 在价格前面加上人民币符号（￥）

```
//第 3 章/局部过滤器.html
<!DOCTYPE html>
```

```
<html lang="en" xmlns:v-bind="http://www.w3.org/1999/xhtml">
<head>
<meta charset="UTF-8">
<title></title>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<div id="app">
<!-- 文本后边需要添加管道符号(|)作为分隔,管道符"|"后边是文本的处理函数,处理函数的第一个参数是管道符前边的文本内容,如果处理函数用于传递参数,则从第二个参数往后依次是传递的参数 -->
    <p>计算机价格:{{price | addPriceIcon}}</p>
</div>
<script>
    var vm = new Vue({
        el:"#app",
        data:{
            price:200
        },
        filters:{
            //处理函数
            addPriceIcon(value){
                console.log(value)//200
                return '¥' + value;
            }
        }
    })
</script>
</body>
</html>
```

传递多个参数的局部过滤器如例 3-10 所示。

【例 3-10】 在局部过滤器中传递多个参数

```
//第3章/在局部过滤器中传递多个参数.html
<!DOCTYPE html>
<html lang="en" xmlns:v-bind="http://www.w3.org/1999/xhtml">
<head>
<meta charset="UTF-8">
<title></title>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<div id="app">
```

```
    <!-- 过滤器接收多个参数 -->
    <span>{{value1|multiple(value2,value3)}}</span>
</div>
<script>
    var vm = new Vue({
            el: '#app',
                data: {
                    msg: 'hello',
                    value1:10,
                    value2:20,
                    value3:30
                },
        //局部过滤器
        filters: {
            'multiple': function (value1, value2, value3) {
                return value1 * value2 * value3
            }
        }
    })
</script>
</body>
</html>
```

全局过滤器在实例中的使用如例 3-11 所示。

【例 3-11】 全局过滤器的使用

```
//第 3 章/全局过滤器.html
<!DOCTYPE html>
<html lang="en" xmlns:v-bind="http://www.w3.org/1999/xhtml">
<head>
<meta charset="UTF-8">
<title></title>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<div id="app">
    <h3>{{viewContent | addNamePrefix}}</h3>
</div>
<script>
    /* addNamePrefix 是过滤器的名字,也是管道符后边的处理函数; value 是参数 */
    Vue.filter("addNamePrefix",(value)=>{
        return "my name is" + value
    })
    var vm = new Vue({
```

```
            el:"#app",
            data:{
                viewContent:"贝西"
            }
        })
    </script>
    </body>
    </html>
```

3.2 实例及选项

Vue 通过构造函数来实例化一个 Vue 对象：var vm=new Vue({})。在实例化时，我们会传入一些选项对象，包含数据选项、属性选项、方法选项、生命周期钩子等常用选项。而且 Vue 的核心是一个响应式的数据绑定系统，建立绑定后，DOM 将和数据保持同步，这样无须手动维护 DOM，就能使代码更加简洁易懂，从而提升效率。

3.2.1 数据选项

一般地，当模板内容较简单时，使用 data 选项配合表达式即可。当涉及复杂逻辑时，则需要用到 methods、computed、watch 等方法。

data 是 Vue 实例的数据对象。Vue 使用递归法将 data 的属性转换为 getter/setter，从而让 data 属性能够响应数据变化，代码如下：

```
<!-- 部分代码省略 -->
<div id = "app">
  {{ message }}
</div>
<script>
var values = {message: 'Hello Vue!'}
var vm = new Vue({
  el: '#app',
  data: values
})
console.log(vm);
</script>
```

Vue 实例创建之后，在控制台输入 vm.$data 即可访问原始数据对象，如图 3-2 所示。

在 script 标签中添加一些输出信息即可查看控制台从而观察 Vue 实例是否代理了 data 对象的所有属性，代码如下：

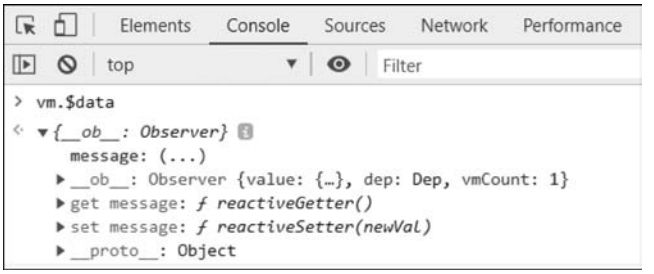

图 3-2　访问原始数据对象

```
<script>
  console.log(vm.$data === values);          //true
  console.log(vm.message);                   //'Hello Vue!'
  console.log(vm.$data.message);             //'Hello Vue!'
</script>
```

被代理的属性是响应式的，也就是说值的任何改变都将触发视图的重新渲染。设置属性也会影响到原始数据，反之亦然。

但是，以“_”或“$”开头的属性不会被 Vue 实例代理，因为它们可能和 Vue 内置的属性或方法冲突。可以使用如 vm.$data._property 的方式访问这些属性，代码如下：

```
<script>
var values = {
  message: 'Hello Vue!',
  _name: 'beixi'
}
var vm = new Vue({
  el: '#app',
  data: values
})
console.log(vm._name);                 //undefined
console.log(vm.$data._name);           //'beixi'
</script>
```

3.2.2　属性选项

Vue 为组件开发提供了属性(props)选项，可以使用它为组件注册动态属性，来处理业务之间的差异性，使代码可以在相似的应用场景复用。

props 选项可以是数组或者对象类型，用于接收从父组件传递过来的参数，并允许为其赋默认值、类型检查和规则校验等，如例 3-12 所示。

【例 3-12】 props 选项的使用

```
//第 3 章/props 选项的使用.Html
<!DOCTYPE html>
<html>
<head>
<title>Hello World</title>
<script src='http://cdnjs.cloudflare.com/ajax/libs/vue/1.0.26/vue.min.js'></script>
</head>
<body>
<div id="app">
    <message content='Hello World'></message>
</div>
</body>
<!-- 测试组件 -->
<script type="text/javascript">
    var Message = Vue.extend({
        props:['content'],
        data:function(){
            return{
            a:'it worked'
        }},
        template:'<h1>{{content}}</h1><h1>{{a}}</h1>'
    })
    Vue.component('message', Message)
    var vm = new Vue({
        el:'#app',
    })
</script>
</html>
```

3.2.3 方法选项

可以通过选项属性 methods 对象来定义方法,并且使用 v-on 指令来监听 DOM 事件,如例 3-13 所示。

【例 3-13】 通过 methods 对象来定义方法

```
//第 3 章/通过 methods 对象来定义方法.html
<!DOCTYPE html>
<html xmlns:v-on="http://www.w3.org/1999/xhtml">
<head>
<title></title>
<meta charset="utf-8"/>
<script src='http://cdnjs.cloudflare.com/ajax/libs/vue/1.0.26/vue.min.js'></script>
```

```
</head>
<body>
<div id="app">
<button v-on:click="test">点我</button>
</div>
</body>
<!-- 测试组件 -->
<script type="text/javascript">
    var vm = new Vue({
        el : '#app',
        methods:{
/*定义了一个test函数*/
            test: function () {
                console.log(new Date().toLocaleTimeString());
            }
        }
    })
</script>
</html>
```

3.2.4 计算属性

在项目开发中，我们展示的数据往往需要处理。除了在模板中绑定表达式或利用过滤器外，Vue还提供了计算属性方法，计算属性是当其依赖属性的值发生变化时，这个属性的值会自动更新，与之相关的DOM部分也会同步自动更新，从而减轻在模板中的业务负担，保证模板的结构清晰和可维护性。

有时候需要在{{}}中进行一些计算再展示出数据，如例3-14所示。

【例3-14】 在页面中展示学生的成绩、总分和平均分

```
//第3章/学生成绩信息.html
<!DOCTYPE html>
<html xmlns:v-on="http://www.w3.org/1999/xhtml">
<head>
<title></title>
<meta charset="utf-8"/>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<div id="app">
<table border="1">
<thead>
    <th>学科</th>
    <th>分数</th>
```

```
</thead>
<tbody>
<tr>
    <td>数学</td>
    <td><input type="text" v-model="Math"></td>
</tr>
<tr>
    <td>英语</td>
    <td><input type="text" v-model="English"></td>
</tr>
<tr>
    <td>语文</td>
    <td><input type="text" v-model="Chinese"></td>
</tr>
<tr>
    <td>总分</td>
    <td>{{Math + English + Chinese}}</td>
</tr>
<tr>
    <td>平均分</td>
    <td>{{(Math + English + Chinese)/3}}</td>
</tr>
</tbody>
</table>
<script>
        var vm = new Vue({
            el:'#app',
            data:{
                Math:66,
                English:77,
                Chinese:88
            }
        })
</script>
</div>
</body>
</html>
```

执行结果如图 3-3 所示。

学科	分数
数学	66
英语	77
语文	88
总分	231
平均分	77

图 3-3 学生成绩、总分和平均分的展示

虽然通过{{ }}运算，可以解决我们的问题，但是代码结构不清晰，特别是当运算比较复杂的时候，我们不能复用功能代码。这时，大家不难想到用 methods 来封装功能代码，但事实上，Vue 提供了一个更好的解决方案——计算属性。计算属性是 Vue 实例中的一个配置选项：computed，通常计算相关的函数，返回最后计算出来的值。也就是我们可以把这些计算的过

程写到一个计算属性中，然后让它动态地计算，如例 3-15 所示。

【例 3-15】 使用计算属性展示学生的成绩、总分和平均分

```
//第 3 章/使用计算属性展示学生成绩.html
<!DOCTYPE html>
<html xmlns:v-on="http://www.w3.org/1999/xhtml">
<head>
<title></title>
<meta charset="utf-8"/>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<div id="app">
<table border="1">
<thead>
    <th>学科</th>
    <th>成绩</th>
</thead>
<tbody>
<tr>
    <td>数学</td>
    <td><input type="text" v-model.number="Math"></td>
</tr>
<tr>
    <td>英语</td>
    <td><input type="text" v-model.number="English"></td>
</tr>
<tr>
    <td>语文</td>
    <td><input type="text" v-model.number="Chinese"></td>
</tr>
<tr>
    <td>总分</td>
    <td>{{sum}}</td>
</tr>
<tr>
    <td>平均分</td>
    <td>{{average}}</td>
</tr>

</tbody>
</table>
</div>
<script>
    var vm = new Vue({
```

```
        el:'#app',
        data:{
            Math:66,
            English: 77,
            Chinese:88
        },
        computed:{
                <!-- 一个计算属性的 getter -->
            sum:function(){
                <!-- this 指向 vm 实例 -->
                return this.Math + this.English + this.Chinese;
            },
                average:function(){
                    return Math.round(this.sum/3);
    }
        }
    });
</script>
</body>
</html>
```

计算属性一般通过其他的数据计算出一个新数据，它的一个好处是能把新的数据缓存下来，当其他的依赖数据没有发生改变时，它调用的是缓存数据，这就极大地提高了程序的性能和数据提取的速度。而如果将计算过程写在 methods 中，数据就不会缓存下来，所以每次都会重新计算。这也是我们没有采用 methods 的原因，如例 3-16 所示。

【例 3-16】 计算属性和方法的比较

```
//第 3 章/计算属性和方法的比较.html
<!DOCTYPE html>
<html xmlns:v-on="http://www.w3.org/1999/xhtml">
<head>
<title></title>
<meta charset="utf-8"/>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<div id="app">
    <p>原始字符串: "{{ message }}"</p>
    <p>计算属性反向字符串: "{{ reversedMessage1 }}"</p>
    <p>methods 方法反向字符串: "{{ reversedMessage2() }}"</p>
</div>
<script>
    var vm = new Vue({
```

```
        el: '#app',
        data: {
            message: 'beixi'
        },
        computed:{
            reversedMessage1: function () {
                return this.message.split('').reverse().join('')
            }
        },
        methods: {
            reversedMessage2: function () {
                return this.message.split('').reverse().join('')
            }
        }
    })
</script>
</body>
</html>
```

计算属性是基于它们的依赖进行缓存的。计算属性只有在它的相关依赖发生改变时才会重新求值。这就意味着只要 message 没有发生改变，多次访问 reversedMessage1 计算属性会立即返回之前的计算结果，而不必再次执行函数。相比而言，只要发生重新渲染，methods 总会调用并执行该函数。

假设有一个性能开销比较大的计算属性 A，它需要遍历一个极大的数组和做大量的计算。可能有其他的计算属性依赖于 A，如果没有缓存，则将不可避免地多次执行 A 的 getter，这样便极大地降低了数据的提取速度，从而导致用户体验感不强。如果不希望有缓存，则用 methods 替代。

实例 3-15 和实例 3-16 只提供了 getter 读取值的方法，实际上除了 getter，还可以设置计算属性的 setter，如例 3-17 所示。

【例 3-17】 读取和设置值

```
//第 3 章/读取和设置值.html
<!DOCTYPE html>
<html xmlns:v-on="http://www.w3.org/1999/xhtml">
<head>
<title></title>
<meta charset="utf-8"/>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<script>
    var vm = new Vue({
```

```
        data: { a: 1 },
        computed: {
            //该函数只能读取数据,使用 vm.aDouble 即可读取数据
            aDouble: function () {
                return this.a * 2
            },
            //读取和设置数据
            aPlus: {
                get: function () {
                    return this.a + 1
                },
                set: function (v) {
                    this.a = v - 1
                }
            }
        }
    })
    console.log(vm.aPlus);        //2
    vm.aPlus = 3;
    console.log(vm.a);            //2
    console.log(vm.aDouble);      //4
</script>
</body>
</html>
```

3.2.5　表单控件

1. 基础用法

可以用 v-model 指令在表单< input >、< textarea >及< select >元素上创建双向数据绑定,如图 3-4 所示。它会根据控件类型自动选取正确的方法来更新元素。尽管有些神奇,但 v-model 本质上不过是语法糖。它负责监听用户的输入事件以便更新数据,并对一些极端场景进行一些特殊处理。

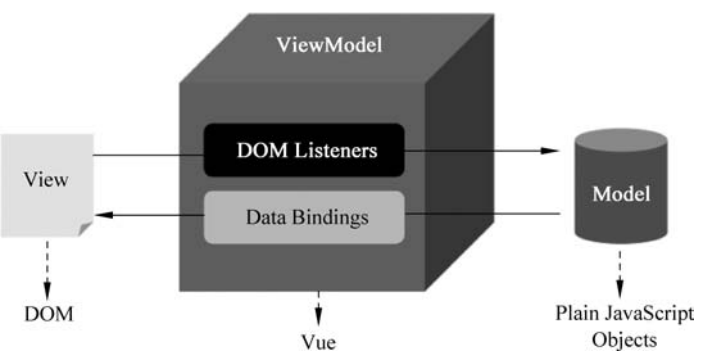

图 3-4　Vue 双向数据绑定

注意：v-model 会忽略所有表单元素的 value、checked、selected attribute 的初始值而总是将 Vue 实例的数据作为数据来源，所以应该通过 JavaScript 在组件的 data 选项中声明初始值。

(1) 单行文本可在 input 元素中使用 v-model 实现双向数据绑定，代码如下：

```
<div id = "app" class = "demo">
    <input v - model = "message" placeholder = "请输入信息...">
    <p>Message is: {{ message }}</p>
</div>
<script>
    var vm = new Vue({
        el: '#app',
        data: {
            message: ''
        }
    })
</script>
```

(2) 多行文本可在 textarea 元素中使用 v-model 实现双向数据绑定，代码如下：

```
<div id = "example - textarea" class = "demo">
    <span>Multiline message is:</span>
    <p style = "white - space: pre">{{ message }}</p>
    <br>
    <textarea v - model = "message" placeholder = "add multiple lines"></textarea>
</div>
<script>
    new Vue({
        el: '#example - textarea',
        data: {
            message: ''
        }
    })
</script>
```

注意：在文本区域插值(<textarea></textarea>)并不会生效，应用 v-model 来代替。

(3) 单个复选框，绑定到布尔值，代码如下：

```
<div id = "example - checkbox" class = "demo">
    <input type = "checkbox" id = "checkbox" v - model = "checked">
    <label for = "checkbox">{{ checked }}</label>
</div>
<script>
    new Vue({
```

```
        el: '#example-checkbox',
        data: {
            checked: false
        }
    })
</script>
```

(4) 多个复选框,绑定到同一个数组,代码如下:

```
<div id="example-checkboxs" class="demo">
    <input type="checkbox" id="jack" value="Jack" v-model="checkedNames">
    <label for="jack">Jack</label>
    <input type="checkbox" id="john" value="John" v-model="checkedNames">
    <label for="john">John</label>
    <input type="checkbox" id="mike" value="Mike" v-model="checkedNames">
    <label for="mike">Mike</label>
    <br>
    <span>Checked names: {{ checkedNames }}</span>
</div>
<script>
    new Vue({
        el: '#example-checkboxs',
        data: {
            checkedNames: []
        }
    })
</script>
```

(5) 单选按钮的双向数据绑定,代码如下:

```
<div id="example-radio">
    <input type="radio" id="runoob" value="Runoob" v-model="picked">
    <label for="runoob">Runoob</label>
    <br>
    <input type="radio" id="google" value="Google" v-model="picked">
    <label for="google">Google</label>
    <br>
    <span>选中值为: {{ picked }}</span>
</div>
<script>
    new Vue({
        el: '#example-radio',
        data: {
            picked : 'Runoob'
```

```
        }
    })
</script>
```

(6) 选择列表,下拉列表的双向数据绑定。

- 单选时,代码如下:

```
<div id = "example - selected">
<select v - model = "selected">
    <option disabled value = "">请选择</option>
    <option>A</option>
    <option>B</option>
    <option>C</option>
</select>
<span>Selected: {{ selected }}</span>
</div>
<script>
    new Vue({
        el: '#example - selected',
        data: {
            selected: ''
        }
    })
</script>
```

注意:如果 v-model 表达式的初始值未能匹配任何选项,则<select>元素将被渲染为"未选中"状态。在 iOS 中,这会使用户无法选择第一个选项。因为在这种情况下,iOS 不会触发 change 事件。因此,更推荐像上面那样提供一个值为空的禁用选项。

- 多选列表(绑定到一个数组),代码如下:

```
<div id = "example - selected" class = "demo">
    <select v - model = "selected" multiple style = "width: 50px">
        <option>A</option>
        <option>B</option>
        <option>C</option>
    </select>
    <br>
    <span>Selected: {{ selected }}</span>
</div>
<script>
    new Vue({
        el: '#example - selected',
        data: {
```

```
            selected: []
        }
    })
</script>
```

- 动态选项，用 v-for 渲染，代码如下：

```
<div id = "example - selected" class = "demo">
    < select v - model = "selected">
        < option v - for = "option in options" v - bind:value = "option.value">
                    {{ option.text }}
        </option>
    </select>
< span > Selected: {{ selected }}</span>
</div>
< script >
    new Vue({
        el: '#example - selected',
        data: {
            selected: 'A',
            options: [
                { text: 'One', value: 'A' },
                { text: 'Two', value: 'B' },
                { text: 'Three', value: 'C' }
                ]
            }
})
</script>
```

2. 绑定 value

(1) 对于单选按钮、勾选框及选择列表选项，v-model 绑定的 value 通常是静态字符串，代码如下：

```
<div id = "app">
    <!-- 当选中时,picked 为字符串 "a" -->
    < input type = "radio" v - model = "picked" value = "a">
    <!-- 当选中时,toggle 为 true 或 false -->
    < input type = "checkbox" v - model = "toggle">
    <!-- 当选中时,selected 为字符串 "abc" -->
    < select v - model = "selected">
        < option value = "abc"> ABC </option>
    </select>
</div>
< script >
```

```
var vm = new Vue({
        el: '#app',
        data: {
            picked:'',
            toggle:'',
            selected:''
        }
    })
</script>
```

(2) 实现复选框功能。有时我们想绑定 value 到 Vue 实例的一个动态属性上,这时可以用 v-bind 实现,并且这个属性的值可以不是字符串,代码如下:

```
<div id = "app">
<input type = "checkbox" v-model = "toggle" v-bind:true-value = "a" v-bind:false-value =
"b">
    {{toggle}}
</div>
<script>
var vm = new Vue({
        el: '#app',
        data: {
            toggle:'',
            a:'a',
            b:'b'
        }
    })
</script>
```

当选中时输出字符串 a;当没有选中时输出字符串 b。

(3) 单选按钮,代码如下:

```
<input type = "radio" v-model = "pick" v-bind:value = "a">
```

当选中时在页面控制台输入 vm.pick 的值和 vm.a 的值相等。

(4) 选列表设置,代码如下:

```
<select v-model = "selected">
    <!-- 内联对象字面量 -->
    <option v-bind:value = "{ number: 123 }">123</option>
</select>
```

当选中时在页面控制台输入 typeof vm.selected,则输出'object';如果输入 vm.selected.number,则输出值为 123。

3. 修饰符

• .lazy

在默认情况下，v-model 每次在 input 事件触发后将输入框的值与数据进行同步（除了上述输入法组合文字时）。此时可以添加.lazy 修饰符，从而转为在 change 事件之后进行同步，代码如下：

```
<!-- 在"change"时而非"input"时更新 -->
<input v-model.lazy="msg">
```

• .number

如果想自动将用户的输入值转为数值类型，可以给 v-model 添加.number 修饰符，代码如下：

```
<input v-model.number="age" type="number">
```

通常这很有用，因为即使在 type="number" 时，HTML 输入元素的值也总会返回字符串。如果这个值无法被 parseFloat()解析，则会返回原始的值。

• .trim

如果要自动过滤用户输入的首尾空白字符，可以给 v-model 添加.trim 修饰符，代码如下：

```
<input v-model.trim="msg">
```

3.2.6 生命周期

Vue 实例有一个完整的生命周期，也就是开始创建、初始化数据、编译模板、挂载 DOM、渲染→更新→渲染、卸载等一系列过程，我们称之为 Vue 的生命周期。通俗地说就是 Vue 实例从创建到销毁的过程，如图 3-5 所示。

可以看到在 Vue 的整个生命周期中有很多钩子函数，这就给了用户在不同阶段添加自己代码的机会，下面先列出所有的钩子函数，然后再一一详解。

(1) beforeCreate：在实例初始化之后，数据观测(data observer)和 event/watcher 事件配置之前被调用。

(2) Created：实例已经创建完成之后被调用。在这一步，实例已完成以下配置，数据观测(data observer)，属性和方法的运算，以及 watch/event 事件回调。然而，挂载阶段还没开始，$el 属性目前不可见。

(3) beforeMount：在挂载开始之前被调用，相关的 render 函数首次被调用。

(4) Mounted：el 被新创建的 vm.$el 替换，并挂载到实例上之后调用该钩子。

(5) beforeUpdate：数据更新时调用，发生在虚拟 DOM 重新渲染和打补丁之前。可以在这个钩子中进一步更改状态，这不会触发附加的重新渲染过程。

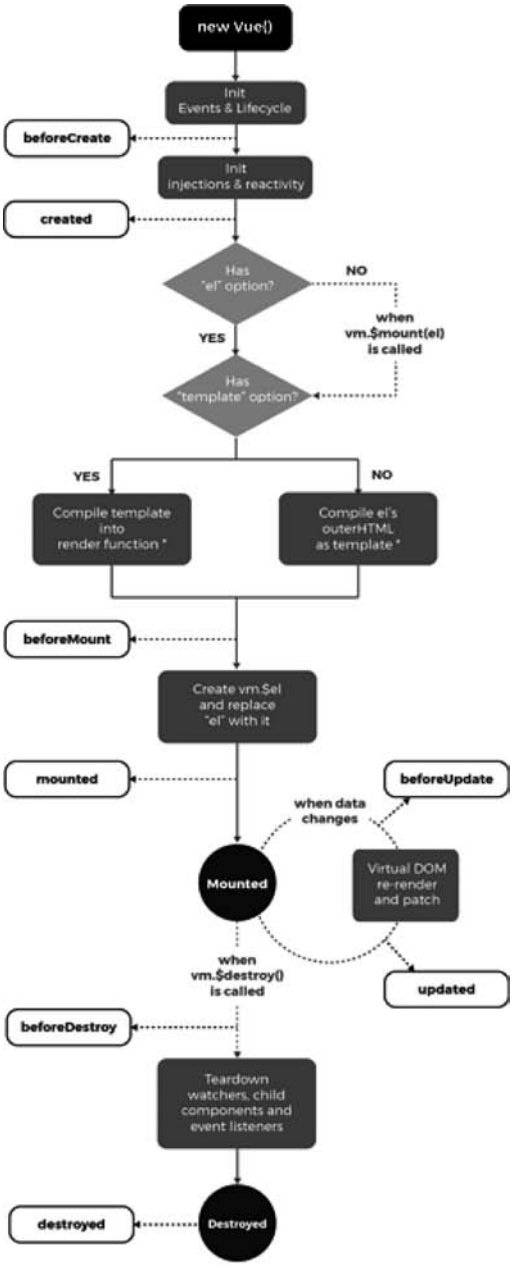
new Vue()
Init
Events & Lifecycle
beforeCreate
Init
injections & reactivity
created
Has
"el" option?
NO
when
vm.$mount(el)
is called
YES
Has
"template" option?
YES
NO
Compile template
into
render function *
Compile el's
outerHTML
as template *
beforeMount
Create vm.$el
and replace
"el" with it
mounted
beforeUpdate
when data
changes
Mounted
Virtual DOM
re-render
and patch
when
vm.$destroy()
is called
updated
beforeDestroy
Teardown
watchers, child
components and
event listeners
destroyed
Destroyed
* template compilation is performed ahead-of-time if using
a build step, e.g. single-file components

图 3-5　Vue 生命周期

(6) Updated：由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子。当这个钩子被调用时，组件 DOM 已经更新，所以现在可以执行依赖于 DOM 的操作。然而在大多数情况下，应该避免在此期间更改状态，因为这可能会导致更新无限循环。该钩子在服务器端渲染期间不会被调用。

(7) beforeDestroy：实例销毁之前调用。在这一步，实例仍然完全可用。

(8) Destroyed：Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会被解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。该钩子在服务器端渲染期间不会被调用。

下面通过实例代码来演示 Vue 实例在创建过程中调用的几个生命周期钩子，如例 3-18 所示。

【例 3-18】 生命周期钩子函数的演示

```
//第 3 章/生命周期钩子函数.html
<!DOCTYPE html>
<html xmlns:v-on="http://www.w3.org/1999/xhtml"
xmlns:v-bind="http://www.w3.org/1999/xhtml">
<head>
<title></title>
<meta charset="utf-8"/>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<div id="app">
<h1>{{message}}</h1>
</div>
</body>
<script>
    var vm = new Vue({
        el: '#app',
        data: {
            message: 'Vue 的生命周期'
        },
        beforeCreate: function() {
            console.group('------ beforeCreate 创建前状态 ------');
            console.log("%c%s", "color:red" , "el : " + this.$el);       //undefined
            console.log("%c%s", "color:red","data : " + this.$data);     //undefined
            console.log("%c%s", "color:red","message: " + this.message)
        },
        created: function() {
            console.group('------ created 创建完毕状态 ------');
            console.log("%c%s", "color:red","el : " + this.$el);         //undefined
            console.log("%c%s", "color:red","data : " + this.$data);     //已被初始化
```

```
        console.log("%c%s","color:red","message: " + this.message);  //已被初始化
    },
    beforeMount: function() {
        console.group('------ beforeMount 挂载前状态 ------');
        console.log("%c%s" ,"color:red","el : " + (this.$el));      //已被初始化
        console.log(this.$el);
        console.log("%c%s", "color:red","data : " + this.$data);    //已被初始化
        console.log("%c%s", "color:red","message: " + this.message); //已被初始化
    },
    mounted: function() {
        console.group('------ mounted 挂载结束状态 ------');
        console.log("%c%s","color:red","el : " + this.$el);         //已被初始化
        console.log(this.$el);
        console.log("%c%s", "color:red","data : " + this.$data);    //已被初始化
        console.log("%c%s", "color:red","message: " + this.message); //已被初始化
    },
    beforeUpdate: function () {
        console.group('beforeUpdate 更新前状态 ===============》');
        console.log("%c%s", "color:red","el : " + this.$el);
        console.log(this.$el);
        console.log("%c%s", "color:red","data : " + this.$data);
        console.log("%c%s", "color:red","message: " + this.message);
    },
    updated: function () {
        console.group('updated 更新完成状态 ===============》');
        console.log("%c%s", "color:red","el : " + this.$el);
        console.log(this.$el);
        console.log("%c%s", "color:red","data : " + this.$data);
        console.log("%c%s", "color:red","message: " + this.message);
    },
    beforeDestroy: function () {
        console.group('beforeDestroy 销毁前状态 ===============》');
        console.log("%c%s", "color:red","el : " + this.$el);
        console.log(this.$el);
        console.log("%c%s", "color:red","data : " + this.$data);
        console.log("%c%s", "color:red","message: " + this.message);
    },
    destroyed: function () {
        console.group('destroyed 销毁完成状态 ===============》');
        console.log("%c%s", "color:red","el : " + this.$el);
```

```
            console.log(this.$el);
            console.log("%c%s", "color:red","data : " + this.$data);
            console.log("%c%s", "color:red","message: " + this.message)
        }
    })
</script>
</html>
```

运行后打开 console 可以看到打印出来的内容，如图 3-6 所示。

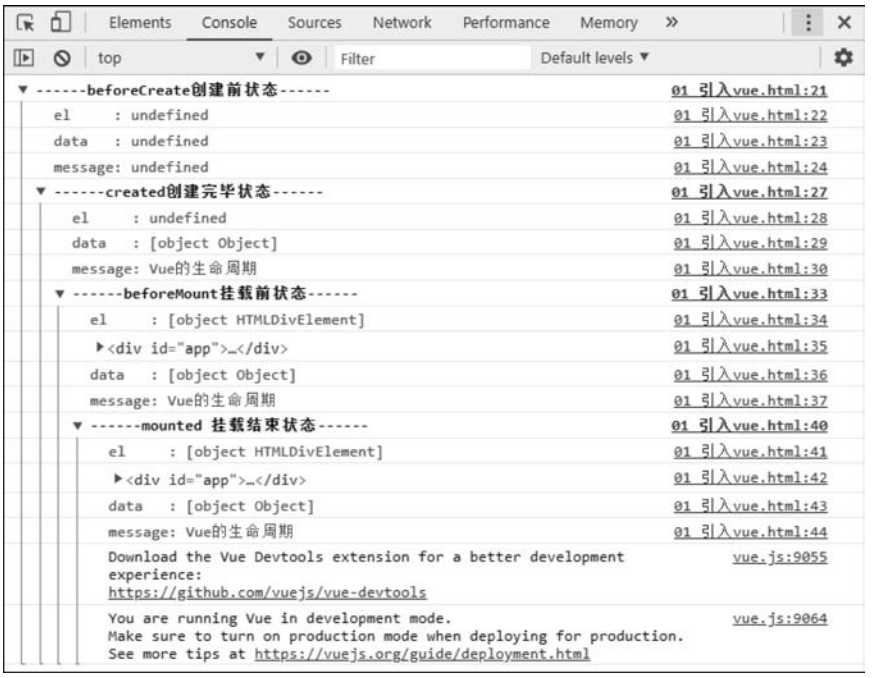

图 3-6　Vue 实例在创建过程中调用的几个生命周期钩子函数

接下来将详细解释生命周期钩子函数。

(1) 在 beforeCreate 和 created 钩子函数之间的生命周期。

在这个生命周期之间，首先进行初始化事件，然后进行数据观测，此时可以看到在 created 的时候数据已经和 data 属性进行了绑定(放在 data 中的属性的值发生改变时，视图也会改变)。

注意：此时还是没有 el 选项。

(2) created 钩子函数和 beforeMount 间的生命周期，如图 3-7 所示。

在这一阶段发生的事情还是比较多的。首先会判断对象是否有 el 选项。如果有就继续向下编译，如果没有 el 选项，则停止编译，也就意味着停止了生命周期，直到在该 Vue 实

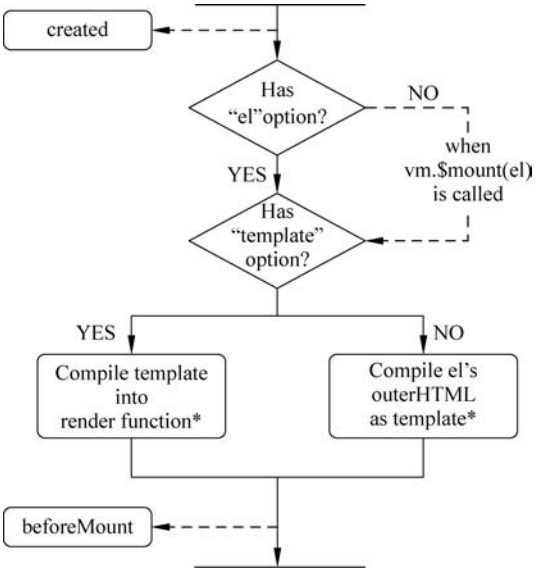

图 3-7　created 和 beforeMount 函数间的生命周期

例上调用 vm. $mount(el)。此时注释掉代码中的:el: '#app',然后可以看到程序运行到 created 时就停止了,如图 3-8 所示。

图 3-8　没有 el 选项的生命周期

如果在后面继续调用 vm. $mount(el),则代码继续向下执行,如图 3-9 所示。

接着往下看,template 参数选项的有无对生命周期的影响,如例 3-19 所示。

(1) 如果 Vue 实例对象中有 template 参数选项,则将其作为模板编译成 render 函数。

(2) 如果没有 template 选项,则将外部 HTML 作为模板编译。

(3) 可以看到 template 中的模板优先级要高于 outer HTML 的优先级。

【例 3-19】 Vue 对象中增加了 template 选项

```
//第 3 章/Vue 对象中增加了 template 选项.html
<!DOCTYPE html>
```

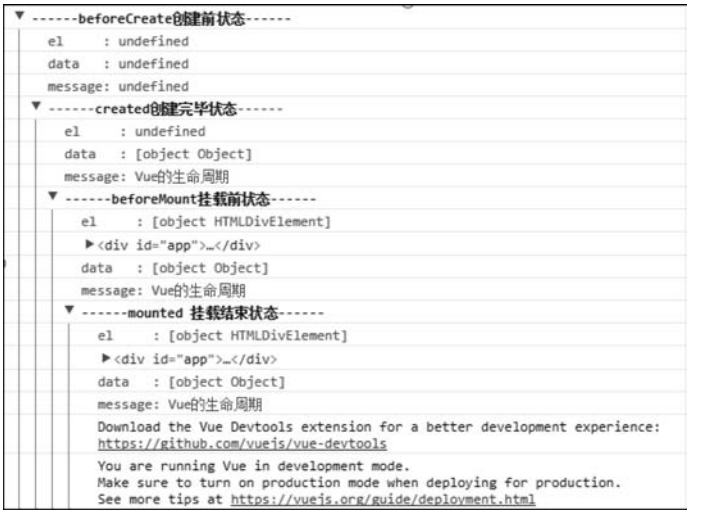

图 3-9 调用 vm.$mount(el)的生命周期

```
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>Vue 生命周期学习</title>
<script src="https://cdn.bootcss.com/vue/2.4.2/vue.js"></script>
</head>
<body>
<div id="app">
<!-- html 中修改的 -->
<h1>{{message + '这是在 outer HTML 中的'}}</h1>
</div>
</body>
<script>
    var vm = new Vue({
        el: '#app',
        //在 Vue 配置项中修改
            template: "<h1>{{message + '这是在 template 中的'}}</h1>",
            data: {
            message: 'Vue 的生命周期'
            }
                })
</script>
</html>
```

运行结果如图 3-10 所示。

图 3-10　在 Vue 对象中增加了 template 选项

将 Vue 对象中 template 的选项注释掉后显示的结果如图 3-11 所示。

图 3-11　在 Vue 对象中去掉 template 选项

注意：el 进行 DOM 绑定要在 template 之前，因为 Vue 需要通过 el 找到对应的 outer template。

在 Vue 对象中还有一个 render 函数，它是以 createElement 作为参数的，然后进行渲染操作，并且我们可以直接嵌入 JSX，代码如下：

```
var vm = new Vue({
    el: '#app',
    render: function(createElement) {
        return createElement('h1', 'this is createElement')
    }
})
```

运行程序后可以看到页面渲染的结果如图 3-12 所示。

图 3-12　render 函数渲染结果

所以综合排名优先级：render 函数选项> template 选项> outer HTML。

(1) beforeMount 和 mounted 钩子函数间的生命周期，如图 3-13 所示。

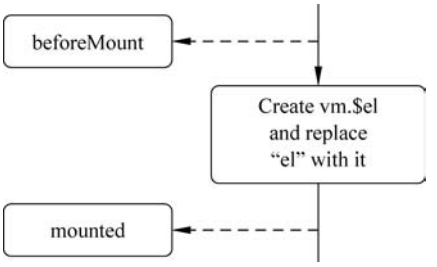

图 3-13　beforeMount 和 mounted 钩子函数间的生命周期

可以看到此时给 Vue 实例对象添加 $el 成员，并且替换掉挂载的 DOM 元素。在之前 console 中打印的结果可以看出 beforeMount 之前 el 的属性还是 undefined。

(2) 接下来讲解 mounted 钩子函数的生命周期。

在 mounted 之前 h1 还是通过{{message}}进行占位的，因为此时还没有挂载到页面上，还是在 JavaScript 中以虚拟 DOM 的形式存在。在 mounted 之后可以看到 h1 的内容发生了变化，如图 3-14 所示。

图 3-14 mounted 函数前后内容变化

(3) beforeUpdate 钩子函数和 updated 钩子函数间的生命周期，如图 3-15 所示。

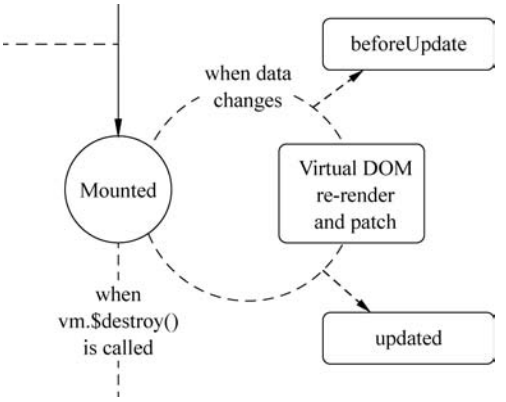

图 3-15 beforeUpdate 和 updated 钩子函数间的生命周期

当 Vue 发现 data 中的数据发生了改变时，会触发对应组件的重新渲染，先后调用 beforeUpdate 和 updated 钩子函数。我们在 console 中输入如下信息：

```
vm.message = '触发组件更新'
```

此时可以发现触发了组件的更新，如图 3-16 所示。

(4) beforeDestroy 和 destroyed 钩子函数间的生命周期，如图 3-17 所示。

beforeDestroy 钩子函数在实例销毁之前调用。在这一步，实例仍然完全可用。而

```
> vm.message = '触发组件更新'
▼ beforeUpdate 更新前状态==============》
    el     : [object HTMLDivElement]
    ▶<div id="app">…</div>
    data   : [object Object]
    message: 触发组件更新
  ▼ beforeUpdate 更新前状态==============》
      el     : [object HTMLDivElement]
      ▶<div id="app">…</div>
      data   : [object Object]
      message: 触发组件更新
    ▼ updated 更新完成状态==============》
        el     : [object HTMLDivElement]
        ▶<div id="app">…</div>
        data   : [object Object]
        message: 触发组件更新
      <· "触发组件更新"
>
```

图 3-16　组件更新状态

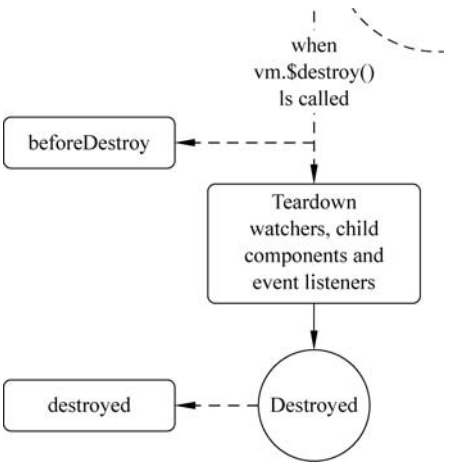

图 3-17　beforeDestroy 和 destroyed 钩子函数间的生命周期

destroyed 钩子函数在 Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会被解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。

3.3　模板渲染

当我们获取后台数据之后，会按照一定的规则加载到前端写好的模板中，并显示在浏览器中，这个过程称为渲染。

3.3.1　条件渲染

1. v-if、v-else-if 和 v-else

v-if、v-else-if、v-else 这 3 个指令后面跟的是表达式。Vue 的条件指令可以根据表达式

的值在 DOM 中渲染或销毁元素/组件，如例 3-20 所示。

【例 3-20】 v-if、v-else-if 、v-else 应用

```
//第 3 章/条件指令.html
<!DOCTYPE html>
<html xmlns:v-on="http://www.w3.org/1999/xhtml"
xmlns:v-bind="http://www.w3.org/1999/xhtml">
<head>
<title></title>
<meta charset="utf-8"/>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<div id="app">
    <!-- if、else 指令 -->
    <p v-if="status==1">当 status 为 1 时，显示该行</p>
    <p v-else-if="status==2">当 status 为 2 时，显示该行</p>
    <p v-else-if="status==3">当 status 为 3 时，显示该行</p>
    <p v-else>否则显示该行</p>
</div>
<!-- script 脚本 -->
<script>
    //创建 Vue 实例
    var vm=new Vue({
        el: '#app',
        data: {
            status: 2
        }
    });
</script>
</body>
</html>
```

我们需要注意多个 v-if、v-else-if 和 v-else 之间需要紧挨着，如果在它们之间插入一行新代码，则代码如下：

```
<p v-if="status==1">当 status 为 1 时，显示该行</p>
<span></span>
<p v-else-if="status==2">当 status 为 2 时，显示该行</p>
<p v-else-if="status==3">当 status 为 3 时，显示该行</p>
<p v-else>否则显示该行</p>
```

此时编译浏览器会报错，如图 3-18 所示。

2. v-show

实际上同 v-if 效果等同，当绑定事件的元素符合引号中的条件时，该元素才显示，代码如下：

图 3-18　v-if、v-else-if 和 v-else 之间不挨着时显示的错误信息

```
<div id = "app">
<!-- if、else 指令 -->
<p v - show = "status == 1">当 status 为 1 时，显示该行</p>
<p v - show = "status == 2">当 status 为 2 时，显示该行</p>
</div>
<script>
    //创建 Vue 实例
    var vm = new Vue({
        el: '#app',
        data: {
            status: 2
        }
    });
</script>
```

3. v-if 和 v-show 的区别

(1) 控制显示的方法不同：该方法和 v-if 区别在于，v-show 实际通过修改 DOM 元素的 display 属性来实现节点的显示和隐藏，而 v-if 则通过添加/删除 DOM 节点来实现。

(2) 编译条件：v-if 是惰性的，如果初始条件为假，则什么也不做，此时不会去渲染该元素，只有在条件第一次变为真时才开始局部编译；v-show 则不管初始条件是什么，都被编译，然后被缓存，而且将 DOM 元素保留，只是简单地基于 CSS 进行切换，即 v-if 条件为 true 时才会被加载渲染，而为 false 时不加载。v-show 不管为 true 还是 false，都会被加载渲染。

(3) 性能消耗：v-if 有更高的切换消耗，而 v-show 有更高的初始渲染消耗。

(4) 使用场景：因此，如果需要非常频繁地切换，则使用 v-show 较好。如果在运行时条件很少改变，只需一次显示或隐藏，则使用 v-if 较好。

4. key

Vue 会尽可能高效地渲染元素，通常会复用已有元素而不是从头开始渲染，如例 3-21 所示，因此可能造成下面这样的问题。

【例 3-21】 Vue 高效地渲染元素

```
//第 3 章/Vue 高效地渲染元素.html
<!DOCTYPE html>
```

```
<html xmlns:v-on="http://www.w3.org/1999/xhtml"
xmlns:v-bind="http://www.w3.org/1999/xhtml">
<head>
<title></title>
<meta charset="utf-8"/>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<div id="app">
    <p v-if="ok">
        <label>Username</label>
        <input placeholder="Enter your username">
    </p>
    <p v-else>
        <label>Email</label>
        <input placeholder="Enter your email address">
    </p>
    <button @click="ok=!ok">切换</button>
</div>
<script type="text/javascript">
    var vm=new Vue({
        el:'#app',
            data:{
                ok:true,
            }
    })
</script>
</body>
</html>
```

页面中输入信息后单击“切换”按钮，此时文本框里的内容并没有清空。

Vue为我们提供了一种方式来声明“这两个元素是完全独立的，不要复用它们”。只需添加一个具有唯一值的key属性，代码如下：

```
<div id="app">
    <p v-if="ok">
        <label>Username</label>
        <input placeholder="Enter your username" key="username-input">
    </p>
    <p v-else>
        <label>Email</label>
        <input placeholder="Enter your email address" key="email-input">
    </p>
    <button @click="ok=!ok">切换</button>
</div>
```

3.3.2 列表渲染

1. v-for 循环用于数组

v-for 指令根据一组数组的选项列表进行渲染。

我们可以用 v-for 指令基于一个数组来渲染一个列表。v-for 指令需要使用 item in items 形式的特殊语法，其中 items 是源数据数组，而 item 则是被迭代的数组元素的别名（为当前遍历的元素提供别名，可以任意起名），如例 3-22 所示。

【例 3-22】 对数组选项进行列表渲染

```
//第 3 章/数组选项列表渲染.html
<!DOCTYPE html>
<html xmlns:v-on="http://www.w3.org/1999/xhtml"
xmlns:v-bind="http://www.w3.org/1999/xhtml">
<head>
<title></title>
<meta charset="utf-8"/>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<ul id="app">
    <li v-for="item in items">
            {{ item.name }}
    </li>
</ul>
<script type="text/javascript">
    var vm = new Vue({
        el: '#app',
            data: {
                items: [
                    { name: 'beixi' },
                    { name: 'jzj' }
                ]
            }
    })
</script>
</body>
</html>
```

我们定义一个数组类型的数据 items，用 v-for 将<li>标签循环渲染，效果如图 3-19 所示。

- beixi
- jzj

图 3-19 列表循环结果

v-for 还支持一个可选的第 2 个参数为当前项的索引，索引从 0 开始，代码如下：

```
<ul id="app">
    <li v-for="(item,index) in items">
            {{index}} -- {{ item.name }}
    </li>
</ul>
```

分隔符 in 前的语句使用括号，第 2 项就是 items 当前项的索引，运行的结果如图 3-20 所示。

图 3-20 含有 index 选项的列表渲染结果

注意：可以用 of 代替 in 作为分隔符。

2. v-for 用于对象

v-for 通过一个对象的属性来遍历并输出结果，如例 3-23 所示。

【例 3-23】 v-for 来遍历对象

```
//第 3 章/v-for 来遍历对象.html
<!DOCTYPE html>
<html xmlns:v-on="http://www.w3.org/1999/xhtml"
xmlns:v-bind="http://www.w3.org/1999/xhtml">
<head>
<title></title>
<meta charset="utf-8"/>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<ul id="app">
    <li v-for="value in object">
            {{ value }}
    </li>
</ul>
<script type="text/javascript">
    var vm = new Vue({
        el: '#app',
            data: {
                object: {
                    name: 'beixi',
                    gender: '男',
                    age: 30
                }
            }
    })
</script>
</body>
</html>
```

渲染后的结果如图 3-21 所示。

遍历对象属性时，有两个可选参数，分别是键名和索引，代码如下：

```
<ul id = "app">
    <li v - for = "(value, key, index) in object">
            {{ index }} -- {{ key }}: {{ value }}
    </li>
</ul>
```

渲染后的结果如图 3-22 所示。

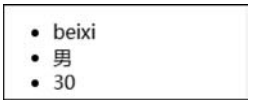

图 3-21　遍历对象结果

图 3-22　遍历对象的渲染结果

3. v-for 用于整数

v-for 还可以迭代整数，代码如下：

```
<ul id = "app">
    <li v - for = "n in 10">
            {{n}}
    </li>
</ul>
<script type = "text/javascript">
        var vm = new Vue({
            el: '#app',
        })
</script>
```

3.3.3　template 标签用法

上述例子中，v-if 和 v-show 指令都包含在一个根元素中，那是否有方法可以将指令作用到多个兄弟 DOM 元素上呢？Vue 提供了内置标签< template >，可以将多个元素进行渲染，代码如下：

```
<div id = "app">
    <div v - if = "ok">
        <p>这是第 1 段代码</p>
        <p>这是第 2 段代码</p>
        <p>这是第 3 段代码</p>
    </div>
</div>
```

```
<script type="text/javascript">
    var vm = new Vue({
        el: '#app',
        data:{
            ok:true
        }
    })
</script>
```

同样,<template>标签也支持使用 v-for 指令,用来渲染同级的多个兄弟元素,代码如下:

```
<ul id="app">
<template v-for="item in items">
        {{ item.name }}
        {{ item.age }}
</template>
</ul>
<script type="text/javascript">
    var vm = new Vue({
        el: '#app',
        data: {
            items: [
                { name: 'beixi' },
                { age: 18 }
            ]
        }
    })
</script>
```

3.4 事件绑定

Vue 提供了 v-on 指令来监听 DOM 事件,在事件绑定上,类似原生 JavaScript 的 onclick 事件写法,也是在 HTML 上进行监听。

3.4.1 基本用法

Vue 中的事件绑定,使用 v-on:事件名=函数名来完成,这里函数名定义在 Vue 实例的 methods 对象中,Vue 实例可以直接访问其中的方法。

语法规则:

```
v-on:事件名.修饰符 = 方法名() | 方法名 | 简单的JS表达式
```

```
简写：@事件名.修饰符 = 方法名() | 方法名 | 简单的 JS 表达式
事件名：click|keydown|keyup|mouseover|mouseout|自定义事件名
```

1. 直接使用

直接在标签中书写 js 方法，代码如下：

```
<div id = "app">
    单击次数{{count}}
    <button @click = "count++">单击 + 1 </button>
</div>
<script type = "text/javascript">
    var vm = new Vue({
        el: '#app',
       data:{
           count:0
       }
    })
</script>
```

注意：@click 等同于 v-on:click，是一个语法糖。

2. 调用 methods 的方法

通过 v-on 绑定实例选项属性 methods 中的方法作为事件的处理器，如例 3-24 所示。

【例 3-24】 v-on 绑定事件

```
//第 3 章/v - on 绑定事件.html
<!DOCTYPE html>
<html xmlns:v - on = "http://www.w3.org/1999/xhtml"
xmlns:v - bind = "http://www.w3.org/1999/xhtml">
<head>
<title></title>
<meta charset = "utf - 8"/>
<script src = "https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<div id = "app">
    <button @click = "say">单击</button>
</div>
<script type = "text/javascript">
    var vm = new Vue({
        el: '#app',
       data:{
           msg:'Say Hello'
       },
```

```
        methods:{
            say: function () {
                alert(this.msg)
            }
        }
    })
</script>
</body>
</html>
```

单击 button 按钮，即可触发 say 函数，弹出 alert 框'Say Hello'。

(1) 方法传参，方法直接在调用时在方法内传入参数，代码如下：

```
<div id = "app">
    <button @click = "say('Hello beixi')">单击</button>
</div>
<script type = "text/javascript">
    var vm = new Vue({
        el: '#app',
       data:{
           msg:'Say Hello'
       },
        methods:{
            say: function (val) {
                alert(val)
            }
        }
    })
</script>
```

(2) 传入事件对象，代码如下：

```
<div id = "app">
<button data - aid = '123' @click = "eventFn( $ event)">事件对象</button>
</div>
<script type = "text/javascript">
    var vm = new Vue({
        el: '#app',
        methods:{
            eventFn:function(e){
                console.log(e);
                // e.srcElement,DOM 节点
                e.srcElement.style.background = 'red';
                console.log(e.srcElement.dataset.aid); /* 获取自定义属性的值 */
```

```
            }
        }
    })
</script>
```

3.4.2 修饰符

Vue为指令v-on提供了多个修饰符，方便我们处理一些DOM事件的细节，Vue主要的修饰符如下。

(1).top：阻止事件继续传播，即阻止它的捕获和冒泡过程。代码如下：

```
@click.stop='handle()'      //只要在事件后面加上 .stop 就可以阻止事件冒泡
```

如例3-25所示单击“内部单击”按钮，阻止了冒泡过程，即只执行inner这个方法，如果不加.stop，则先执行inner方法，然后执行outer方法，即通过了冒泡这个过程。

【例3-25】 .stop修饰符应用

```
//第3章/stop修饰符应用.html
<!DOCTYPE html>
<html xmlns:v-on="http://www.w3.org/1999/xhtml"
xmlns:v-bind="http://www.w3.org/1999/xhtml">
<head>
<title></title>
<meta charset="utf-8"/>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
<div id="app">
    <div style="background-color: aqua;width: 100px;height: 100px" v-on:click="outer">
        外部单击
        <div style="background-color: red;width: 50px;height: 50px" v-on:click.stop=
        "inner">内部单击</div>
    </div>
</div>
<script type="text/javascript">
    var vm = new Vue({
        el: '#app',
        methods:{
            outer: function () {
                console.log("外部单击")
            },
            inner: function () {
```

```
                console.log("内部单击")
            }
        }
    })
</script>
</body>
</html>
```

(2).prevent：阻止默认事件。代码如下：

```
@click.prevent = 'handle()'     //只要在事件后面加上.prevent 就可以阻止默认事件
```

如下阻止了 a 标签的默认刷新：

```
<a href = "" v-on:click.prevent>单击</a>
```

(3).capture：添加事件监听器时使用事件捕获模式，即在捕获模式下触发。代码如下：

```
@click.capture = 'handle()'
```

如下实例在单击最里层的“单击 6”时，outer 方法先执行，因为 outer 方法在捕获模式执行，先于冒泡事件。按下列执行顺序执行：outer→set→inner，因为后两个还是冒泡模式下触发的事件，代码如下：

```
<div v-on:click.capture = "outer">外部单击 5
    <div v-on:click = "inner">内部单击 5
        <div v-on:click = "set">单击 6 </div>
    </div>
</div>
```

(4).self：当前元素自身是触发处理函数时才会触发函数。

原理是根据 event.target 确定是否为当前元素本身，以此来决定是否触发事件/函数。

如下示例，如果单击“内部单击”按钮，冒泡不会执行 outer 方法，因为 event.target 指的是内部单击的 DOM 元素，而不是外部单击的，所以不会触发自己的单击事件，代码如下：

```
<div v-on:click.self = "outer">
   外部单击
    <div v-on:click = "inner">内部单击</div>
</div>
```

（5）.once：只触发一次。代码如下：

```
<div id = "app">
    <div v - on:click.once = "once">单击 once</div>
</div>
<script type = "text/javascript">
    var vm = new Vue({
        el: '#app',
        methods:{
            once: function () {
                console.log("单击 once")
            }
        }
    })
</script>
```

（6）键盘事件。

方式一：@keydown='show($event)'

```
<div id = "app">
    <input type = "text" @keydown = 'show( $event)' />
</div>
<script type = "text/javascript">
    var vm = new Vue({
        el: '#app',
        methods:{
            show: function (ev) {
                /* 在函数中获取 ev.keyCode */
                console.log(ev.keyCode);
                if(ev.keyCode == 13){
                    alert('你按了回车键!')
                }
            }
        }
    })
</script>
```

方式二：

```
<input type = "text" @keyup.enter = "show()">        //回车执行
<input type = "text" @keydown.up = 'show()'>         //上键执行
<input type = "text" @keydown.down = 'show()'>       //下键执行
<input type = "text" @keydown.left = 'show()'>       //左键执行
<input type = "text" @keydown.right = 'show()'>      //右键执行
```

3.5 基础 demo 案例

在前面几个章节中，我们介绍了一些 Vue 的基础知识，结合以上知识我们可以做个小 demo：图书管理系统。图书管理系统主要实现数据的添加、删除、列表渲染等功能，最终效果如图 3-23 所示。

图 3-23 图书管理系统效果图

这个 demo 是基于 BootStrap 快速搭建的，所以对 BootStrap 不是很了解的同学，可以先自行到官网 http://www.bootcss.com/进行学习。开始 demo 之前需下载 bootstrap 文件，这里采用的版本是 bootstrap-3.3.7.css。

3.5.1 列表渲染

```
<!DOCTYPE html>
<html xmlns:v-on="http://www.w3.org/1999/xhtml"
xmlns:v-bind="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <meta charset="utf-8"/>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <!-- 引入 bootstrap -->
    <link rel="stylesheet" href="./lib/bootstrap-3.3.7.css">
</head>
<body>
<div class="app">
    <div class="panel panel-primary">
        <div class="panel-heading">
            <h2>图书管理系统</h2>
        </div>
        <div class="panel-body form-inline">
            <label for="">id: <input type="text" class="form-control" v-model="id">
</label>
```

```
            <label for="">图书名称:<input type="text" class="form-control" v-model="name"></label>
            <input type="button" value="添加" class="btn btn-primary" @click="add">
        </div>
    </div>
    <table class="table table-bordered table-hover">
        <thead>
        <tr>
            <th>id</th>
            <th>图书名称</th>
            <th>添加时间</th>
            <th>操作</th>
        </tr>
        </thead>
        <tbody>
        <tr v-for="item in arr" :key="item.id">
            <td v-text="item.id"></td>
            <td v-text="item.name"></td>
            <td v-text="item.time"></td>
            <td><a href="" @click.prevent="del(item.id)">删除</a></td>
        </tr>
        </tbody>
    </table>
</div>
<script>
    var vm = new Vue({      //创建 Vue 实例
        el:'.app',
            data:{
                arr:[
                    {'id':1,'name':'三国演义','time':new Date()},
                    {'id':2,'name':'红楼梦','time':new Date()},
                    {'id':3,'name':'西游记','time':new Date()},
                    {'id':4,'name':'水浒传','time':new Date()}
            ],          //创建一些初始数据及其格式
                id:'',
                name:''
            },
    })
</script>
</body>
</html>
```

3.5.2 功能实现

添加功能的代码如下：

```
add(){
        this.arr.push({'id':this.id,'name':this.name,'time':new Date()});
         this.id = this.name = '';
    }
```

删除功能的代码如下：

```
del(id){
       var index = this.arr.findIndex(item => {
        if(item.id == id) {
                  return true;
               }
          })
      //findIndex 方法查找索引,实现列表的删除功能
      this.arr.splice(index,1)
}
```

完整示例代码如下：

```
<!DOCTYPE html>
<html xmlns:v-on="http://www.w3.org/1999/xhtml"
xmlns:v-bind="http://www.w3.org/1999/xhtml">
<head>
<meta charset="utf-8"/>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<link rel="stylesheet" href="./lib/bootstrap-3.3.7.css">
</head>
<body>
<div class="app">
    <div class="panel panel-primary">
        <div class="panel-heading">
            <h2>图书管理系统</h2>
        </div>
        <div class="panel-body form-inline">
            <label for="">id:<input type="text" class="form-control" v-model="id">
</label>
            <label for="">图书名称:<input type="text" class="form-control" v-model=
"name"></label>
            <input type="button" value="添加" class="btn btn-primary" @click="add">
        </div>
    </div>
    <table class="table table-bordered table-hover">
        <thead>
        <tr>
            <th>id</th>
```

```
                <th>图书名称</th>
                <th>添加时间</th>
                <th>操作</th>
            </tr>
            </thead>
            <tbody>
                <tr v-for="item in arr" :key="item.id">
                <td v-text="item.id"></td>
                <td v-text="item.name"></td>
                <td v-text="item.time"></td>
                <td><a href="" @click.prevent="del(item.id)">删除</a></td>
            </tr>
            </tbody>
        </table>
    </div>
    <script>
        var vm = new Vue({      //创建 Vue 实例
            el:'.app',
                data:{
                    arr:[
                        {'id':1,'name':'三国演义','time':new Date()},
                        {'id':2,'name':'红楼梦','time':new Date()},
                        {'id':3,'name':'西游记','time':new Date()},
                        {'id':4,'name':'水浒传','time':new Date()}
                    ],              //创建一些初始数据与格式
                    id:'',
                    name:''
                },
                methods:{
                    add(){
                        this.arr.push({'id':this.id,'name':this.name,'time':new Date()});
                        this.id = this.name = '';
                    },          //add 方法实现列表的输入功能
                    del(id){
                        var index = this.arr.findIndex(item => {
                            if(item.id == id) {
                                return true;
                            }
                        })
    /* findIndex 方法查找索引,实现列表的删除功能 */
                        this.arr.splice(index,1)
                    }
                }
        })
    </script>
    </body>
    </html>
```