项目三 Vue指令

本章内容

- 3.1 Vue指令概述
- 3.2 基本指令
- 3.3 条件渲染
- 3.4 列表渲染
- 3.5 自定义指令

3.1 Vue指令概述

- 3.1.1 指令
- 3.1.2 参数
- 3.1.3 指令修饰符

3.1.1 指令

定义

- 指令是带有 v- 前缀的特殊属性
- 指令用于在表达式的值改变时，将某些行为应用到 DOM 上。
- 作用：当表达式的值改变时，将其产生的连带影响，响应式的作用于 DOM。

例3.1 指令示例

```
<div id="app">
    <p v-if="seen">现在你看到我了</p>
    <template v-if="ok">
      <h1>学习使人进步</h1>
    </template>
</div>
<script>
new Vue({
        el: '#app',
        data: {
          seen: true,
          ok: true
        }
})
</script>
```

3.1.2 参数

- 参数在指令后以冒号指明。例如， v-bind 指令被用来响应地更新 HTML 属性：
- 例3.2 指令参数示例

```
<div id="app">
    <pre><a v-bind:href= “url” >四川托普学院</a></pre>
</div>
<script>
new Vue({
        el: '#app',
        data: {
            url: 'http://www.scetop.com'
        }
})
</script>
```

3.1.3 指令修饰符

- 定义
  - 指令的修饰符使用“.”指明的特殊后缀，表示指令应该以特殊的方式绑定
- 常见修饰符
  - .prevent 修饰符会阻止当前事件的默认行为
  - .stop 修饰符将阻止事件向上冒泡

例3.3 指令修饰符示例

```
<div id='app' class="row">
        <h2>v-on.prevent</h2>
        <form @submit.prevent="form_submit"
action="http://www.baidu.com/index.html">
            <button type="submit">Submit</button>
        </form>
        <hr />
    </div>
    <script>
        var vm = new Vue({
            el: "#app",
            methods: {
                form_submit: function () {
                    alert("form submit!");
                }
            }
        });
    </script>
```

3.2 Vue基本指令

- 3.2.1 v-cloak
- 3.2.2 v-once
- 3.2.3 v-text与v-html
- 3.2.4 v-bind
- 3.2.5 v-on

3.2.1 v-cloak

- 在使用Vue的过程中，当引入了vue.js这个文件之后，浏览器的内存中就存在了一个Vue对象，我们可以通过构造函数的方式创建出一个Vue的对象实例，后面就可以对这个实例进行操作。

- 如果在这个过程中，对于vue.js的引用因为某些原因没有加载完成，此时，未编译的Mustache标签就无法正常显示。例如，在下面的例子中，我们模拟将网页加载速度变慢，此时就可以看见，页面最先开始会显示出插值表达式，只有vue.js加载完成后，才会渲染成正确的数据。

例3.4 v-cloak示例

```
<div id="app" >
    <p>{{message}}</p>
</div>
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<script>
    new Vue({
        el: '#app',
        data: {
            message: 'hello world!'
        }
    });
</script>
```

例3.4 v-cloak示例

- 在浏览器运行可发现，vue.js加载完成后，才会渲染成正确的数据，文件还没加载完时，在页面上会显示{{message}}的字样，直到Vue创建实例、编译模板时，DOM才会被替换，这个过程屏幕是有闪动的。
- 解决问题：
-

```
<style>
        [v-cloak]{
            display: none;
        }
    </style>
```

```
<!-- 注：使用 v-cloak 能够解决 插值表达式闪烁的问题 -->
```

3.2.2 v-once

- v-once指令只渲染元素和组件一次，随后的渲染，使用了此指令的元素、组件及其所有的子节点，都会当作静态内容并跳过，这可以用于优化更新性能。

例3.5 v-once示例

```
<div id="app">
        <p >未加v-once：{{msg}}</p>
        <p v-once>原始值：{{msg}}</p>
        <p>改变后的值：{{msg}}</p>
        <p><input type="text" v-model = "msg" name=""></p>
    </div>
    <script>
        new Vue({
            el: '#app',
            data: {
                msg : "hello"
            }
        })
    </script>
```

执行一次性地插值，当数据改变时，插值处的内容不会更新。

3.2.3 v-text

- v-text: 设置标签的文本值(textContent)　例3.6 v-text示例

```
<div id="app">
        <h2 v-text="message+'!'">学校</h2>
        <h2 v-text="address+'!'">地址</h2>
        <h2>{{ message +'!'}}高新西区</h2>
    </div>
    <script>
        var app = new Vue({
            el:"#app",
            data:{
                message:"四川托普信息技术职业学院",
                address:"四川省成都市高新区西区大道2000号"
            }
        })
    </script>
```

3.2.3 v-text

总结：

- ◆ v-text指令的作用是:设置标签的内容(textContent)
- 默认写法会替换全部内容,使用插值表达式{{}}可以替换指定内容
- 内部支持写表达式

3.2.4 v-html

- v-html: 设置标签的innerHTML　例3.7 v-html示例

```
<div id='app'>
      <h1>{{ msg }}</h1>
      <h2 v-text='msg'></h2>
      <div v-html='htmlMsg'></div>
  </div>
  <script>
      // v-html既能插入值 又能插入标签 innerHTML
      new Vue({
          el:'#app',
          data:{
              msg:"插入标签",
              htmlMsg:'<h3>v-html指令应用</h3>'
          }
      })
  </script>
```

v-text与v-html的区别

- v-html :
- v-text与v-html指令都可以更新页面元素的内容，不同的是，v-text会将数据以**字符串文本**的形式更新，而v-html则是将数据以**HTML标签**的形式更新。
- 在更新数据上，我们也可以使用插值表达式进行更新数据，不同于v-text、v-html指令，**插值表达式只会更新原本占位插值所在的数据内容，而v-text、v-html指令则会替换掉整个的内容。**

3.2.4 v-html

- **总结**
  - v-html指令的作用是:设置元素的innerHTML
  - 内容中有html结构会被解析为标签
  - v-text指令无论内容是什么,只会解析为文本
  - 解析文本使用v-text,需要解析html结构使用v-html

3.2.5 v-on

- v-on 为元素绑定事件
- v-on 指令用于绑定 HTML的单击事件用 v-on:click 缩写为 @click。

3.2.5 v-on

例3.8 v-on示例

```
<div id="app">
    <!-- 'greet' 是在下面定义的方法名 -->
    <button v-on:click="greet">Greet</button>
</div>
<script>
    var app = new Vue({
        el: '#app',
        data: {
            name: 'Vue.js'
        },
        methods: {
            greet: function () {
                alert(this.name + '的方法被调用了!');
            }
        }
    })
</script>
```

3.2.5 v-on

总结

- v-on指令的作用是:为元素绑定事件
- 事件名不需要写on
- 指令可以简写为@
- 绑定的方法定义在methods属性中
- 方法内部通过this关键字可以访问定义在data中数据

3.2.6 v-bind

- v-bind可以用来在标签上绑定标签的属性（例如：img的src、title属性等等）和样式（可以用style的形式进行内联样式的绑定，也可以通过指定 class 的形式指定样式）。同时，对于绑定的内容，是做为一个JavaScript变量，因此，可以对该内容进行编写合法的JavaScript表达式。

3.2.6 v-bind

- v-bind 指令属性
  - Vue 指令以 v-前缀开始，数据绑定的指令 v-bind:属性名

- 基本语法 **v-bind:属性名=表达式**

3.2.6 v-bind

- 例3.9 v-bind示例：

```
<h1>绑定标签的指定属性，采用表达式方式</h1>
    <div id="app-2">
        <span v-bind:title= “message” >百度</span>
        <a v-bind:href="url">点击跳转</a>
    </div>
    <script>
        var app = new Vue({
            el: '#app-2',
            data: {
                message: ' 页面加载于 ' + new Date(),
                url: 'http://www.baidu.com/'
            }
        })
    </script>
```

3.2.6 v-bind

- 总结
  - 完整写法是 v-bind:属性名
  - 简写的话可以直接省略v-bind,只保留 :属性名
  - 需要动态的增删class建议使用对象的方式

3.2.7 v-bind绑定HTML样式

- 数组语法
  - Vue中提供了使用数组进行绑定样式的方式，可以直接在数组中写上样式的类名。
  - 注意：如果不使用单引号包裹类名，其实代表的还是一个变量的名称，会出现错误信息。

例3.10 class数组语法

```
<style>
        .static{
            color: white; }
        .style1{
            background: #4f43ff;
        }
        .style2{
            width: 200px;
            height: 100px;
        }
    </style>
```

```
<div id="app">
    <div class="static" v-bind:class="['style1','style2']">
            {{message}}
    </div>
</div>
<script>
    new Vue({
        el: '#app',
        data:{
            message:"数组语法"
        }
    })
</script>
```

3.2.7 v-bind绑定HTML样式

- 对象语法
  - 在Vue中也可以直接使用对象的形式来设置样式。对象的属性为样式的类名，value则为true或者false，当值为true时显示样式。
  - 由于对象的属性可以带引号，也可不带引号，所以属性就按照自己的习惯写法就可以了。

例3.11 class对象语法

```
<style>
        .static{ color: white; }
        .style1{ background: #4f43ff;  }
        .style2{ width: 200px; height: 100px;   }
    </style>
<div id="app">
    <div class="static" v-bind:class="classObject">
            {{message}}
    </div>
</div>
<script>
    new Vue({
        el: '#app',
        data: {
            message:'对象语法',
            boole1: true,
            boole2: true
        },
        computed: {
            classObject: function () {
                return {
                    style1:this.boole1,
                    'style2':this.boole2
                }
            }
        }
    })
</script>
```

3.2.8 v-bind绑定内联样式

- 内联样式是将CSS样式编写到元素的style属性中。
- 数组语法
  - v-bind:style的数组语法可以将多个样式对象应用到同一个元素上，样式对象可以是data中定义的样式对象和计算属性中return的对象。

例3.12 style数组语法示例

```
<div id="app">
    <div v-bind:style="[styleObject1,styleObject2]">数组语法</div>
</div>
<script>
    new Vue({
        el: '#app',
        data: {
            styleObject1: {
                color: 'blue',
                fontSize: '30px'
            }
        },
        //计算属性
        computed:{
            styleObject2:function(){
                return {
                    border: '1px solid red',
                    padding: '30px'
                }
            }
        }
    })
</script>
```

3.2.8 v-bind绑定内联样式

- 对象语法
  - 与使用属性为元素设置class样式相同，在Vue中，也可以使用对象的方式，为元素设置style样式。
  - v-bind:style的对象语法十分直观——看着非常像CSS，但其实是一个JavaScript对象。**CSS属性名可以用驼峰式（camelCase）或短横线分隔（kebab-case，记得用引号包裹起来）来命名。**

例3.13 style对象语法示例

```
<div id="app">
    <div v-bind:style="styleObject">对象语法</div>
</div>
<script>
    new Vue({
        el: '#app',
        //计算属性
        computed:{
            styleObject:function(){
                return {
                    color: 'blue',
                    fontSize: '30px'
                }
            }
        }
    })
</script>
```

3.3 条件渲染

- 3.3.1 v-if
- 3.3.2 在<template>元素上使用v-if条件渲染分组
- 3.3.3 v-else
- 3.3.4 v-else-if
- 3.3.5 用key管理可复用的元素
- 3.3.6 v-show
- 3.3.7 v-if与v-show

3.3.1 v-if

- v-if指令用于条件性地渲染一块内容。这块内容只会在指令的表达式返回truthy(真)值的时候被渲染。

例3.11 v-if指令示例

```
<div id="app">
    欢迎大家学习
    <h3 v-if="value">Vue.js</h3>
    <h3 v-if="!value">Angular.js</h3>
</div>
<script>
    new Vue({
        el: '#app',
        data: {
            value:true,
        }
    })
</script>
```

```
<div id="app">
欢迎大家学习
<h3 v-if="!value">Vue.js</h3>
<h3 v-else>Angular.js</h3>
</div>
<script>
new Vue({
el: '#app',
data: {
value:true,
}
})
</script>
```

3.3.2 在<template>元素上使用v-if条件渲染分组

- 因为v-if是一个指令，所以必须将它添加到一个元素上。但是如果想切换多个元素，此时可以把一个<template>元素当做不可见的包裹元素，并在上面使用v-if，最终的渲染结果将不包含<template>元素。

例3.12 使用template包裹元素

```
<div id="app">
    <template v-if="value">
        <h3>欢迎大家学习</h3>
        <p>Vue.js</p>
        <p>Angular.js</p>
    </template>
</div>
<script>
    new Vue({
        el: '#app',
        data: {
            value:true,
        }
    })
</script>
```

- 注：template目前不支持v-show，但支持v-if

3.3.3 v-else

- 可以使用v-else指令来表示v-if的“else块”，类似于JavaScript中的if...else逻辑语句。

- v-else元素必须紧跟在带v-if或者v-else-if元素的后面，否则将不会被识别。

例3.13 v-else指令示例

```
<div id="app">
        <h3>value此时的值:{{value}}</h3>
        <div v-if="value>0.5">      <!--如果value>0.5-->
              你现在可以看到我
        </div>
        <div v-else>     <!--否则-->
           你现在看不到我
        </div>
</div>
<script>
new Vue({
        el: '#app',
        data: {
              value:Math.random() //定义一个随机值
        }
})
</script>
```

3.3.4 v-else-if

- v-else-if指令类似于条件语句中的“else-if块”，可以与v-if连续使用。

例3.14 v-else-if指令示例

```
<div id="app">
    <div v-if="type === 'A'">
        A
    </div>
    <div v-else-if="type === 'B'">
        B
    </div>
    <div v-else-if="type === 'C'">
        C
    </div>
    <div v-else>
        Not A/B/C
    </div>
</div>
<script>
    new Vue({
        el: '#app',
        data: {
            type:'C',
        }
    })
</script>
```

3.3.5 用key管理可复用的元素

- 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。这么做除了使Vue变得非常快之外，还有其它一些好处。例如，允许用户在不同的登录方式之间切换。

例3.15 不添加key属性示例

```
<div id="app">
    <template v-if="loginType === 'username'">
        <label>姓名：</label>
        <input placeholder="输入用户名">
    </template>
    <template v-else>
        <label>邮箱：</label>
        <input placeholder="输入邮箱">
    </template>
    <button @click="toggleLoginType">切换</button>
</div>
<script>
    new Vue({
        el: '#app',
        data: {
            loginType: 'username'
        },
        methods: {
            toggleLoginType: function() {
                return this.loginType = this.loginType === 'username' ? 'email' : 'username'
            }
        }
    });
</script>
```

例3.16 添加key属性示例

```
<div id="app">
    <template v-if="loginType === 'username'">
        <label>姓名：</label>
        <input placeholder="输入用户名" key="value1">
    </template>
    <template v-else>
        <label>邮箱：</label>
        <input placeholder="输入邮箱" key="value2">
    </template>
    <button @click="toggleLoginType">切换</button>
</div>
<script>
    new Vue({
        el: '#app',
        data: {
            loginType: 'username'
        },
        methods: {
            toggleLoginType: function() {
                return this.loginType = this.loginType === 'username' ? 'email' : 'username'
            }
        }
    });
</script>
```

3.3.6 v-show

- 另一个用于根据条件展示元素的选项是v-show指令，用法与v-if大致一样，不同的是带有v-show的元素始终会被渲染并保留在DOM中。
- v-show只是简单地切换元素的CSS属性display，当模板属性为true的时候，控制台显示为display:block；属性值为false的时候，控制台显示display:none。

例3.17 v-show指令示例

```
<div id="app">
        <p v-show="ok">v-show控制display属性</p>
        <button v-on:click='toggle()'>toggle</button>
        <p>ok: {{ok}}</p>
</div>
<script>
        var vm = new Vue({
                el:'#app',
                data: {
                        ok:true
                },
                methods: {
                        toggle() {
                                this.ok=!this.ok;
                        }
                }
        })
</script>
```

3.3.7 v-if与v-show

- v-if与v-show指令都是根据表达式的真假值判断元素的显示与隐藏。
- v-if是“真正”的条件渲染，因为它会确保在切换过程中，条件块内的事件监听器和子组件适当地被销毁和重建。
- v-if也是惰性的：如果在初始渲染时条件为假，则什么也不做，直到条件第一次变为真时，才会开始渲染条件块。
- 相比之下，v-show就简单得多，不管初始条件是什么，元素总是会被渲染，并且只是简单地基于CSS进行切换。
- 一般来说，v-if有更高的切换开销，而v-show有更高的初始渲染开销。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，则使用v-if较好。

例3.18 v-show与v-if指令示例

```
<div id="app-2">
        <span v-show="computedSeen">
                v-show:用css(display:none)来显示or隐藏。这不会修改DOM.
        </span>No Show.
        <hr>
        <span v-if="seen">鼠标悬停几秒钟查看此处动态绑定的提示信息！这会修改DOM。</span>
        <h1 v-else>No seen</h1>
        <template v-if="seen">
                <h1>多标记if</h1>
                <p>段落1</p>
                <p>段落2</p>
        </template>
</div>
<button onclick="app.seen = !app.seen;">点击切换</button>
<script src="../js/vue.js"></script>
<script>
        var app = new Vue({
                el:'#app-2',
                data: {
                        seen:true
                },
                computed: {
                        computedSeen: function() {
                                // body...
                                return !this.seen ;
                        }
                }
        })
</script>
```

3.4 列表渲染

- 3.4.1 使用v-for指令遍历元素
- 3.4.2 维护状态
- 3.4.3 数组更新检测
- 3.4.4 在<template>上使用v-for
- 3.4.5 v-for与v-if一同使用

3.4.1 使用v-for指令遍历元素

- 不管是写C#、JAVA还是前端的JavaScript脚本，提到循环数据，首先都会想到使用for循环。同样的，在Vue中，也为我们提供了v-for指令用来循环数据。
- 在使用v-for指令时，可以对数组、对象、数字、字符串进行循环，来获取到源数据中的每一个值。使用v-for指令，必须使用特定语法 alias in expression，其中items是源数据数组，而item则是被迭代的数组元素的别名，具体格式如下：
- ** **
- **{{ item.text }}**
- ** **
- **items是源数据数组， item 是数组元素别名**

3.4.1 使用v-for指令遍历元素

- 通过一个对象的属性来迭代

```
<ul id="repeat-object" class="demo">
 <li v-for="value in object">
{{ value }}
 </li>
</ul>
```

3.4.1 使用v-for指令遍历元素

- 提供第二个的参数为键名

```
<div v-for="(value, key) in object">
 {{ key }} : {{ value }}
</div>
```

- 提供第三个参数为索引

```
<div v-for="(value, key, index) in object">
{{ index }}. {{ key }} : {{ value }}
</div>
```

- 整数迭代

```
<div>
<span v-for="n in 10">{{ n }} </span>
</div>
```

- v-for Template

如同 v-if 模板,渲染一个包含多个元素的块，使用 v-for 遍历多个标签，就需要用 <template> 。

例3.19 使用v-for遍历数组

```
<div id="app">
    <h3>评选出今年最火的几个人：</h3>
    <ul>
        <li v-for="item in items">
            {{ item }}
        </li>
    </ul>
</div>
<script>
    new Vue({
        el: '#app',
        data: {
            items: ['马云','马化腾','蔡徐坤','刘强东']
        }
    })
</script>
```

例3.20 使用v-for指令第二参数

```
<div id="app">
    <h3>评选出今年最火的几个人：</h3>
    <ul>
        <li v-for="(item,index) in items">
            {{ index }}-{{ item }}
        </li>
    </ul>
</div>
<script>
    new Vue({
        el: '#app',
        data: {
            items: ['马云','马化腾','蔡徐坤','刘强东']
        }
    })
</script>
```

例3.21 使用v-for遍历对象

```
<div id="app">
    <h3>人物介绍</h3>
    <ul>
        <li v-for="(value,name,index) in object">
            {{index}}--{{name}}:{{ value }}
        </li>
    </ul>
</div>
<script>
    new Vue({
        el: '#app',
        data: {
            //定义对象
            object: {
                姓名: '蔡徐坤',
                性别: '男',
                出生日期: '2019-10-10'
            }
        }
    })
</script>
```

在遍历对象时，会按Object.keys( )的结果遍历，但是不能保证它的结果在不同的JS引擎下都一致。 Object.keys( ) 用来获取对象自身可枚举的属性键。

3.4.2 维护状态

- 当 Vue 正在更新使用 v-for 渲染的元素列表时，它默认使用“就地更新”的策略。如果数据项的顺序被改变，Vue 将不会移动 DOM 元素来匹配数据项的顺序，而是就地更新每个元素，并且确保它们在每个索引位置正确渲染。这个类似 Vue 1.x 的 track-by="$index"。
- 这个默认的模式是高效的，但是只适用于不依赖子组件状态或临时 DOM 状态 (例如：表单输入值) 的列表渲染输出。
- 为了给 Vue 一个提示，以便它能跟踪每个节点的身份，从而重用和重新排序现有元素，你需要为每项提供一个唯一 key 属性：
-
- <!--内容-->
-
- 建议尽可能在使用v-for时提供key属性，除非遍历输出的DOM内容非常简单，或者是刻意依赖默认行为以获取性能上的提升。

3.4.3 数组更新检测

- Vue为了增加列表渲染的功能，增加了一组观察数组的方法，而且可以显示一个数组的过滤或排序的副本。
- **1.变异方法(mutation method)**
- **2.替换数组**

3.4.3 数组更新检测

- **1.变异方法(mutation method)**

Vue 包含一组观察数组的变异方法，变异方法(mutation method)，顾名思义，会改变被这些方法调用的原始数组

所以它们也将会触发视图更新。这些方法如下：

3.4.3 数组更新检测

- push() 接收任意数量的参数，把它们逐个添加到数组末尾，并返回修改后数组的长度
- pop() 从数组末尾移除最后一项，减少数组的length值，然后返回移除的项
- shift() 移除数组中的第一个项并返回该项，同时数组的长度减1
- unshift() 在数组前端添加任意个项并返回新数组长度
- splice() 删除原数组的一部分成员，并可以在被删除的位置添加入新的数组成员
- sort() 调用每个数组项的toString()方法，然后比较得到的字符串排序，返回经过排序之后的数组
- reverse() 用于反转数组的顺序，返回经过排序之后的数组

例3.22 变异方法示例

```
<div id="example">
  <div>
    <button @click="push()">push</button>
    <button @click="pop()">pop</button>
    <button @click="shift()">shift</button>
    <button @click="unshift()">unshift</button>
    <button @click="splice()">splice</button>
    <button @click="sort()">sort</button>
    <button @click="reverse()">reverse</button>
  </div>
  <ul>
    <li v-for="item in items">
      {{item.message}}
    </li>

  </ul>
</div>
```

例3.22 变异方法示例

```
<script>
 var example = new Vue({
   el:"#example",
   data:{
     items:[
        {message:5},
        {message:2},
        {message:7}
     ],
     addValue:{message:5},
     addSplice:{message:'Thank'},
   },
```

```
methods:{
        push(){
          this.items.push(this.addValue) //末尾添加
        },
        pop(){
          this.items.pop() // 末尾删除
        },
        shift(){
          this.items.shift()   // 开头删除
        },
        unshift(){
          this.items.unshift(this.addValue) //开头添加
        },
        splice(){// 从第二个位置添加一个Thank
          this.items.splice(1,0,this.addSplice); },
        sort(){
          this.items.sort(function(a, b){
             return a.message < b.message;    // 比较大小
          });
        },
        reverse(){
          this.items.reverse()         // 反转数组
        },
      }
  })
</script>
```

3.4.3 数组更新检测

- **2.替换数组**

变异方法，顾名思义，会改变调用了这些方法的原始数组。相比之下，也有非变异 (non-mutating method) 方法，例如 filter()、concat() 和 slice() 。它们不会改变原始数组，而**总是返回一个新数组**。当使用非变异方法时，可以用新数组替换旧数组：

例3.23 filter方法示例

```
<div id="app">
    <ul>
        <li v-for="n in items">{{ n }}</li>
    </ul>
</div>
<script>
    var app=new Vue({
        el:"#app",
        data:{
            numbers: [ 1, 2, 3, 4, 5 ]
        },
        computed:{
            items: function () {
                return this.numbers.filter(function (number)
                {
                    return number<4
                })
            }
        }
    })
</script>
```

- 以上示例操作并不会导致Vue丢弃现有DOM并重新渲染整个列表。Vue实现了一些智能启发式方法来最大化DOM元素重用，所以用一个含有相同元素的数组去替换原来的数组是非常高效的操作

3.4.4 在<template>上使用v-for

- 类似于v-if，可以利用带有v-for的<template>来循环渲染一个包含多个元素的内容。
- <template>在实际渲染的时候元素是**不显示**在网页上的，只是起到一个包裹作用。

例3.24 v-for与template示例

```
<h1>v-for指令根据数组内容渲染列表页</h1>
<div id="myApp">
        <h2>单位基本情况</h2>
        <template v-for="(v,k) in school">
                {{k}} -- {{v}} <br>
        </template>
        单位名称: {{school.name}}
        地址: {{school.address}}
        <h2>{{name}}</h2>
        <ol>
                <template v-for="(i,m) in list">
                        <li>姓名: {{i.message}} 名次: {{m+1}}</li>
                        <p>{{i.mem}}</p>
                </template>
        </ol>
</div>
```

```
<script>
var data={
                    school: {
                            name:'乐美无限',
                            address:'北京'
                    },
                    name: '名单列表',
                    list: [
                            {message: "张三",mem: "该同志是一个好同志！ "},
                            {message: "张三",mem: "该同志是一个好同志！ "},
                            {message: "张三",mem: "该同志是一个好同志！ "},
                            {message: "张三",mem: "该同志是一个好同志！ "},
                            {message: "张三",mem: "该同志是一个好同志！ "},
                            {message: "张三",mem: "该同志是一个好同志！ "},
                            {message: "张三",mem: "该同志是一个好同志！ "},
                            {message: "张三",mem: "该同志是一个好同志！ "},
                            {message: "张三",mem: "该同志是一个好同志！ "},
                            {message: "张三",mem: "该同志是一个好同志！ "}
                    ]
        }
                    var app = new Vue({
                            el:'#myApp',
                            data:data
                    })

</script>
```

3.4.5 v-for与v-if一同使用

- 当它们处于同一节点上时，v-for的优先级比v-if更高，这意味着v-if将分别重复运行于每个v-for循环中。当只想为部分项渲染节点时，这种优先级的机制会十分有用。例如下面示例，循环出没有报道的学生名字。

例3.25 v-for与v-if一同使用

```
<div id="app">
    <h3>没有报道的学生名单：</h3>
    <ul>
        <li v-for="n in items" v-if="!n.value">
            {{ n.name}}
        </li>
    </ul>
</div>
<script>
    var app = new Vue({
        el:'#app',
        data:{
            items:[
                {name:'小明',},
                {name:'小红', value:'已报到'},
                {name:'小华', value:'已报到'},
                {name:'小思'}
            ]
        }
    })
</script>
```

不推荐在同一元素上使用v-if和v-for，

必要情况下应该替换成计算属性

例3.26 替换成计算属性

```
<div id="app">
    <h3>没有报道的学生名单：</h3>
    <ul>
        <li v-for="n in student">
            {{ n.name}}
        </li>
    </ul>
</div>
<script>
    var app = new Vue({
        el:'#app',
        data:{
            items:[
                {name:'小明',},
                {name:'小红',},
                {name:'小华', value:'已报到'},
                {name:'小思'}
            ]
        },
        computed:{
            student:function(){
                return this.items.filter(function (n) {
                    return !n.value
                })
            }
        }
    })
</script>
```

本章小结

- 学习vue指令v-if条件指令、v-for迭代指令、v-show指令
- v-bind绑定vue实例的动态属性
- 使用v-model创建双向数据绑定