第 3 章 Vue 的内置指令与语法

本章将会对 Vue 3.0 中的语法和内置指令进行介绍，包括插值绑定、计算属性、条件渲染指令、列表渲染指令等，对于在 Vue 3.0 中有变更的部分会在对应的小节中进行说明。


视频讲解

3.1 插值绑定

3.1.1 文本插值与表达式

文本插值最基本的方法是使用双大括号(Mustache 语法)“{{ }}”，Vue 将会获取计算后的值，将大括号里的内容替换为设定值，然后以文本的形式将其展示出来。无论通过任何方法修改数据设定值，大括号的内容都会被实时替换。例 2-2 中的 hello、例 2-5 中的 a、b、c 都是通过这种方式在页面中显示数据的。

除了直接赋值，Mustache 语法也接受表达式形式的值。表达式可由 JavaScript 表达式和过滤器构成。表达式可以有变量、数值、运算符等，表达式的值是它的运算结果。虽然不支持条件语句，但可以通过三元式实现简单的条件判断。

例 3-1 展示了通过文本插值与表达式计算变量、表达式、条件运算符的值，在页面中的效果如图 3-1 所示。

【例 3-1】 通过文本插值计算变量、表达式、条件运算符的值

```
<template>
  <div id="app">
    <p><label>变量:</label> {{ num }}</p>
    <p><label>表达式:</label> {{ 5 + 10 }}</p>
    <p><label>条件运算符:</label> {{ true ? 5 : 10 }}</p>
  </div>
</template>

<script>
import { ref } from "vue";

export default {
  setup() {
    const num = ref("2");
```

```
        return {
          num,
        };
      },
    };
    </script></body>
    </html>
```

Vue 的“{{ }}”内只支持单个表达式，不支持语句和流控制。并且在表达式中，不能使用用户自定义的全局变量。“{{ }}”可以放在 HTML 标签内，但 Vue 指令和自身特性内是不可以插值的。

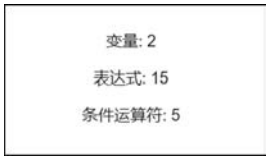


图 3-1 通过文本插值显示变量、表达式、条件运算符值的效果

如果想显示“{{ }}”标签而不进行替换，可以使用 v-pre 跳过这个元素和它的子元素的编译过程。此外，HTML 绑定 Vue 实例，在页面加载时可能会闪烁。原因是 Vue 来不及渲染，页面显示出了 Vue 源代码，可以使用 v-cloak 指令隐藏未编译的 Mustache 标签直到实例准备完毕。

3.1.2 过滤器

在 Vue 2.x 中支持在“{{ }}”插值的尾部添加过滤器，用管道符“|”表示。经常用于格式化文本，如字母全部大写、格式化日期等。过滤的规则是可以自定义的，通过给 Vue 实例添加 filters 来设置。在 Vue 3.0 中，过滤器已被移除，建议使用方法或计算属性来实现。

下面的例 3-2 实现了内置过滤器、过滤器串联与过滤器传参。uppercase 是 Vue 的一个内置过滤器，可以将字符串转换为大写。通过使用多个管道符号可以将多个过滤器进行串联。

【例 3-2】 内置过滤器、过滤器串联与过滤器传参

```
{{ string | uppercase }}
{{ string | filterA | filterB }}
{{ string | filter arg1 arg2 }}
```

当有多个参数时，可以通过空格将参数分开，过滤器会将 string 作为第一个参数，arg1、arg2 分别作为第二个、第三个参数传入。参数可以是表达式，也可以使用单引号传入字符串。

包括 uppercase，Vue 总共内置了 10 种过滤器，将会在第 5 章进行详细介绍。

3.1.3 HTML 插值

HTML 插值可以动态渲染 DOM 节点，常用于处理开发者不可预知和难以控制的 DOM 结构。与文本插值不同的是，文本插值中的代码被解释为节点的文本内容，而 HTML 插值中的代码则被渲染为视图节点。

对于值是 HTML 的片段，可以使用三个大括号“{{{ }}}”来绑定，也可以在标签内使用

v-html=" "的形式。所接收的字符串不会进行编译等操作，Vue 会把被绑定的内容解析为 DOM 节点，按照普通 HTML 处理，从而实现动态渲染视图的效果。

需要注意的是，在网站上直接动态渲染任意 HTML 片段，容易导致 XSS(Cross Site Scripting，跨站脚本攻击)。因此，开发者应尽量多地使用 Vue 自身的模板机制，减少对 HTML 插值的使用，并且只对可信内容使用 HTML 插值。

3.2 计 算 属 性

在项目开发中，往往会在模板中使用表达式或过滤器来对数据进行处理。当表达式过长或者逻辑更复杂时，模板就会变得难以维护。为了避免这种问题，Vue 提供了计算属性，对逻辑进行简化。

3.2.1 计算属性的概念

计算属性会在其依赖属性的值发生变化时，对属性的值进行自动更新，同时更新相关的 DOM 部分。通过从 Vue 中导入 computed 来使用计算属性。

例 3-3 给出了计算属性的例子，double 会始终保持为 num 的两倍，使用按钮增加 num 的值，double 也会随之改变。显示结果如图 3-2 所示。在 Vue Devtools 中可以查看计算属性的值，如图 3-3 所示。

【例 3-3】 计算属性

```
<template>
  <div id="app">
    <p>{{ num }}</p>
    <p>{{ double }}</p>
    <button @click="addBtn">num + 1</button>
  </div>
</template>

<script>
import { computed, toRefs, reactive, ref } from "vue";

export default {
  setup() {
    const data = reactive({
      num: 1,
    });

    const double = computed(() => {
      return data.num * 2;
    });

    const addBtn = () => {
      data.num++;
    };
```

```
    return {
      ...toRefs(data),
      double,
      addBtn,
    };
  },
};
</script>
```

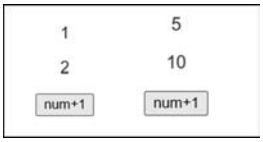


图 3-2　计算属性的显示

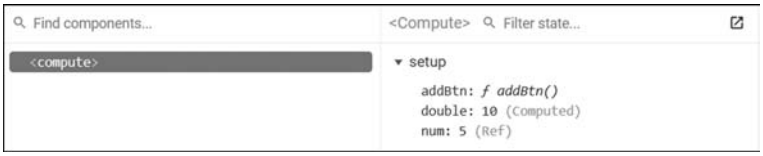


图 3-3　在 Vue Devtools 中查看属性

除了以上这种写法，也可以将 computed 写在 data 中，同样可以达到相同的效果，如例 3-4 所示。

【例 3-4】 计算属性的另一种写法

```
const data = reactive({
    num: 1,
    doubleNum: computed(() => data.num * 2),
  });
```

3.2.2　计算属性

在例 3-3 中，如果尝试修改 double 的值，会发现 Vue 会给出一个 warning，原因是所创建的 double 是只读的计算属性。通过传入一个包含 get 和 set 函数的对象，可以得到一个可读可写的计算属性。

在例 3-5 中创建了一个可读可写的计算属性 tripple，它的值会保持为 num 的三倍。单击按钮将 tripple 的值修改为 9，此时 num 的值也会被一同修改为 3。修改前后的结果如图 3-4 所示。

(a) 修改前　(b) 修改后

图 3-4　修改 tripple 前后的结果

【例 3-5】 可读可写的计算属性

```
const tripple = computed({
  get: () => data.num * 3,
  set: (val) => {
    data.num = val / 3;
  },
});
```

在 computed 中，get 是取值函数，set 是赋值函数。为计算属性赋值时，会触发 set 函数，触发 set 函数后，num 的值会被更新。

3.2.3 侦听属性

watch 函数用来监视指定数据项的变化，从而触发用户自定的操作。watch API 完全等同于选项式 API this. $ watch。watch 需要指定侦听的数据源，并在回调函数中执行副作用。默认情况下，回调仅在侦听的数据源发生改变时调用。通过从 Vue 中导入 watch 函数来使用侦听属性。

例 3-6 给出了使用 watch 函数侦听 reactive 类型的 num 与 ref 类型的 count 的例子。watch 可以获取新值与更新前的值，当 num 与 count 改变时，会执行回调函数，在控制台打印更新前后的值。页面布局与控制台输出结果分别如图 3-5 与图 3-6 所示。

【例 3-6】 侦听 num 与 count 并打印变化

```
//侦听 reactive 类型的数据源
const data = reactive({
  num: 1,
});

watch(
  () => data.num,
  (newNum, oldNum) => {
    console.log("newNum = ", newNum);
    console.log("oldNum = ", oldNum);
  }
);

//侦听 ref 对象
const count = ref(0);

watch(count, (newCount, oldCount) => {
  console.log("newCount = ", newCount);
  console.log("oldCount = ", oldCount);
});
```

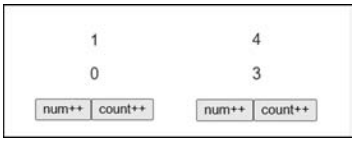


图 3-5 页面布局

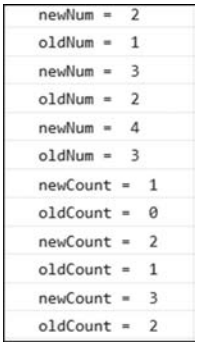


图 3-6 控制台输出结果

侦听器还可以使用数组同时侦听多个源，如例 3-7 所示。

【例 3-7】 侦听多个数据源

```
//侦听多个 reactive 类型
watch(
  [() => data.num1, () => data.num2],
  ([newNum1, newNum2], [prevNum1, prevNum2]) => {
    //Do something
  }
);

//侦听多个 ref 类型
watch(
  [count1, count2],
  ([newCount1, newCount2], [prevCount1, prevCount2]) => {
     //Do something
  }
);
```

3.3 v-bind 属性绑定

除了文本之外，DOM 节点还有一些其他重要的属性，这些属性基本都可以用指令 v-bind 进行绑定。

3.3.1 v-bind 指令

v-bind 指令主要用于动态绑定 DOM 元素属性，可以将一个或多个 attribute，或一个组件 prop 动态地绑定到表达式，如例 3-8 所示。

【例 3-8】 v-bind 示例

```
<!-- 绑定 attribute -->
<img v-bind:src="imageSrc" />
<!-- 缩写 -->
<img :src="imageSrc" />
<!-- 内联字符串拼接 -->
<img :src="'/path/to/images/' + fileName" />

<!-- 动态 attribute 名 -->
<button v-bind:[key]="value"></button>
<!-- 动态 attribute 名缩写 -->
<button :[key]="value"></button>
```

3.3.2 绑定 class、style 与 prop

v-bind 在绑定 class 或 style 的 attribute 时，支持其他类型的值，如数组或对象。虽然类名 class 和样式 style 可接收的类型都是字符串，但类名实际上是由数组拼接而成的，而样式则是由对象键值对拼接而成的。

在绑定 prop 时，prop 必须在子组件中声明。可以用修饰符指定不同的绑定类型，如

例 3-9 所示。

【例 3-9】 绑定 class、style、prop 的 attribute

```
<!-- class 绑定 -->
<div :class="[classA, { classB: isB, classC: isC }]">

<!-- style 绑定 -->
<div :style="{ fontSize: size + 'px' }"></div>
<div :style="[styleObjectA, styleObjectB]"></div>

<!-- prop 绑定."prop" 必须在 my-component 声明 -->
<my-component :prop="someThing"></my-component>

<!-- 通过 $props 将父组件的 props 一起传给子组件 -->
<child-component v-bind="$props"></child-component>
</div>
```

其中,类名 classB 与 classC 分别依赖于数据 isB 和 isC,当 isB 和 isC 为 true 时,div 会拥有类名 classB 与 classC;反之则没有。另外,可以使用三元表达式来根据条件切换类名。

使用:style 时,CSS 属性名称使用驼峰命名(camelCase)或短横分隔命名(kebab-case)。Vue 会自动给特殊的 CSS 属性名称增加前缀,比如 transform。

3.4 v-model 双向绑定

表单控件在项目中十分常用,如输入框、选择等,用它们可以完成数据的录入、提交等操作。通过使用指令 v-model 可以完成表单的数据双向绑定。

3.4.1 v-model 指令

v-model 指令用于在< input >、< textarea >及< select >等表单控件元素上创建双向绑定,它会根据控件类型自动选取正确的方法来更新元素。在修改表单元素值时,对应的实例中的属性值会被同时更新;反之,更改实例中的属性值,表单元素值也会被更新。

在例 3-10 中,给出了用 v-model 绑定< input >、< textarea >及< select >的简单例子,当表单中元素变化时,会更新属性值与视图。元素值更新前后如图 3-7 所示。

【例 3-10】 用 v-model 绑定< input >、< textarea >及< select >

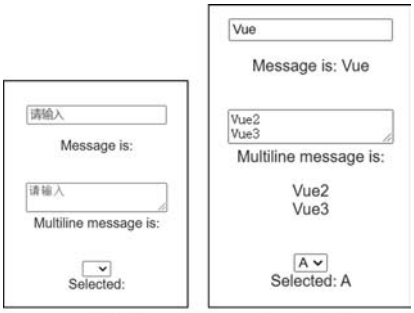
```
<!-- 输入 -->
<input v-model="message" placeholder="请输入" />
<p>Message is: {{ message }}</p>
<br />

<!-- 多行文本 -->
<textarea v-model="mulMessage" placeholder="请输入" ></textarea>
<br />
<span>Multiline message is:</span>
<p style="white-space: pre-line">{{ mulMessage }}</p>
```

```
<br />

<!-- 选择 -->
<select v-model="selected">
  <option>A</option>
  <option>B</option>
  <option>C</option>
</select>
<br />
<span>Selected: {{ selected }}</span>
```

需要注意的是，在文本区域< textarea >中插值不起作用，应该使用 v-model 来代替。

另外，由于< select >的视图太差，而当前也不允许开发者自定义 option 的样式，所以一般都会使用其他元素来模拟下拉选择框。

在例 3-10 中，v-model 绑定的值是一个字符串或布尔值。当需要绑定一个动态的数据时，可以用 v-bind 来实现。



(a) 更新前　(b) 更新后

图 3-7　元素值更新前后

3.4.2　v-model 与修饰符

v-model 的修饰符可以用于控制数据同步的时机。各个修饰符的功能如表 3-1 所示。

表 3-1　v-model 的修饰符

修 饰 符	功　　能
.lazy	将用户输入的数据赋值于变量的时机延迟到数据改变时
.number	将用户输入转换为数值类型
.tmn	删除用户输入的首尾空白字符

在输入框中，v-model 默认是在每次 input 事件触发后将输入框的值与数据进行同步。使用修饰符.lazy 会转变为在 change 事件中同步。数据不是实时改变的，而是在失焦或按 Enter 键时才更新。在例 3-10 的< input >中加入.lazy，在输入框内输入时，message 不会立刻更新并显示，而是在按 Enter 键，或是单击页面其他部分时更新 message。

使用修饰符.number 可以将输入转换为 Number 类型，默认情况下会将输入当作 String 类型。

3.4.3　双向绑定实例：制作问卷

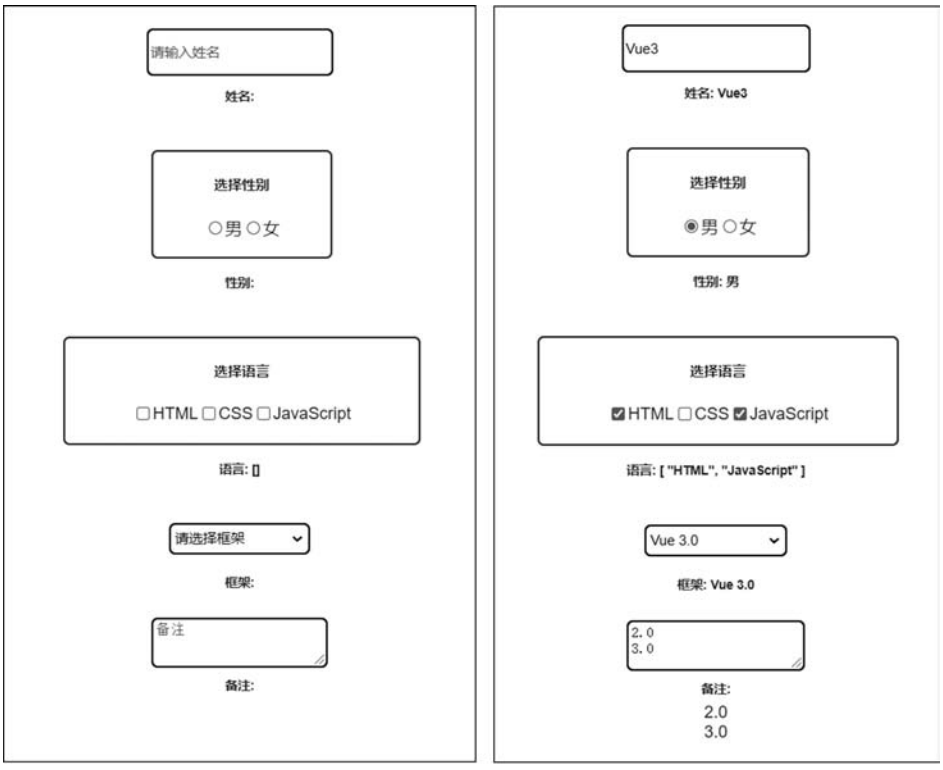
学习了使用 v-model 进行双向绑定，现在可以制作一份实时显示各个表单控件元素中所填数据的问卷。通过指定< input >中的 type，可以实现输入、单选、多选等。

通过问卷采集姓名、性别、编程语言、框架以及备注信息，代码如例 3-11 所示，问卷采集前后的效果如图 3-8 所示。

【例 3-11】 问卷

```
<template>
  <div id="app">
    <input v-model.trim="name" placeholder="请输入姓名" class="inText" />
    <h6>姓名：{{ name }}</h6>
    <br />

    <div class="sex">
      <h5>选择性别</h5>
      <input type="radio" id="男" value="男" v-model="sex" />
      <label for="男">男</label>
      <input type="radio" id="女" value="女" v-model="sex" />
      <label for="女">女</label>
    </div>
    <h6>性别：{{ sex }}</h6>
    <br />

    <div class="mulChoice">
      <h5>选择语言</h5>
      <input type="checkbox" id="HTML" value="HTML" v-model="checkedLang" />
      <label for="HTML">HTML</label>
      <input type="checkbox" id="CSS" value="CSS" v-model="checkedLang" />
      <label for="CSS">CSS</label>
      <input
        type="checkbox"
        id="JavaScript"
        value="JavaScript"
        v-model="checkedLang"
      />
      <label for="JavaScript">JavaScript</label>
    </div>
    <h6>语言：{{ checkedLang }}</h6>
    <br />

    <select v-model="selected" class="select">
      <option disabled selected value style="display: none">请选择框架</option>
      <option>Vue 2.x.x</option>
      <option>Vue 3.0.0</option>
    </select>
    <h6>框架：{{ selected }}</h6>

    <textarea
      v-model="mulMessage"
      placeholder="备注"
      class="mulText"
    ></textarea>
    <br />
    <h6>备注：</h6>
    <p style="white-space: pre-line;margin-top: -20px">{{ mulMessage }}</p>
```

```
      <br />
    </div>
  </template>

  <script>
  import { reactive, ref, toRefs } from "vue";
  export default {
    setup() {
      const data = reactive({
        name: "",
        mulMessage: "",
        selected: "",
        sex: "",
        checkedLang: [],
      });

      return {
        ...toRefs(data),
      };
    },
  };
  </script>
```



(a) 采集前　　(b) 采集后

图 3-8　问卷采集前后的效果

可以看到，当每次在表单控件中输入信息时，数据会实时地显示在页面中，而不需要对页面进行刷新操作。打开 Vue Devtools，可以看到具体的数据内容以及对应的类型，每次对问卷中信息的修改也会直接反映在数据上，如图 3-9 所示。

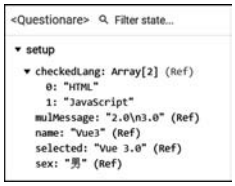


图 3-9　在 Vue Devtools 中查看问卷数据

3.5　v-if/v-show 条件渲染

类似于其他程序中的条件语句 if、else、else if，Vue 的条件指令同样可以根据表达式的值，在 DOM 中渲染或销毁元素与组件，称为条件渲染。

3.5.1　v-if、v-else-if 与 v-else 指令

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真值的时候被渲染。

v-else-if 用于充当 v-if 的“else-if 块”，要紧跟 v-if。当 v-if 中的表达式返回值为假时，开始判断 v-else-if 的表达式值，并根据返回值真假进行渲染。v-else-if 可以连续使用。

v-else 用来表示 v-if 的“else 块”，要紧跟 v-else-if 或 v-if，当 v-else-if 与 v-if 的表达式值均为假时，渲染 v-else 中的内容。

在例 3-12 中使用了 v-if、v-else-if 与 v-else 进行条件渲染，当 type 的值为'A'时，在页面中显示'A'，为'B'时显示'B'，为'C'时显示'C'，其他情况下显示'Not A/B/C'。

【例 3-12】 根据 type 进行条件渲染

```
<template>
  <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>
</template>

<script>
import { reactive, toRefs } from "vue";
export default {
  setup() {
    const data = reactive({
      type: 'A',
    });

    return {
      ...toRefs(data),
    };
```

```
  },
};
</script>
```


其中，在进行相等的条件判断时，应该使用===。===是 Ecmascript 的语言，表示严格等于判断，由于 JavaScript 中的==有缺陷，因此在绝大多数情况下，都是使用三个等号的形式。

因为 v-if 是一个指令，所以必须将它添加到一个元素上。因此，如果想要一次判断多个元素，可以把一个 Vue 内置的< template >元素当作不可见的包裹元素，在上面使用 v-if。最终的渲染结果将不包含< template >元素。如例 3-13 所示，Content A、B、C 将会由同一个 v-if 指令决定是否渲染。

【例 3-13】 在 < template >上使用条件渲染

```
<template v-if="true">
  <h1>Content A</h1>
  <h1>Content B</h1>
  <h1>Content C</h1>
</template>
```

3.5.2 v-show 指令

v-show 指令同样可以用于根据条件展示元素，用法与 v-if 基本相同。不同的是带有 v-show 的元素始终会被渲染并保留在 DOM 中，因为 v-show 只是简单地切换元素 CSS 属性的 display。当条件判定为假时，元素的 display 将被赋值为 none；反之，元素的 display 将被设置为原有值。另外，v-show 不支持< template >元素，也不支持 v-else。

3.5.3 v-if 对比 v-show 指令

v-if 和 v-show 指令具有类似的功能，不过 v-if 才是真正的条件渲染，它会根据表达式适当地销毁或重建元素及绑定的事件或子组件。若表达式初始值为 false，则元素或组件开始并不会渲染，只有当条件第一次变为真时才开始编译。

而 v-show 指令只是简单的 CSS 属性切换，无论条件真与否，都会被编译。元素或组件保留在 DOM 中。

根据各自的特性不难发现，v-if 指令更适合条件不经常改变的场景，因为它切换开销相对较大，而 v-show 指令适用于频繁切换条件。

3.5.4 条件渲染实例：按钮权限控制

在实际的网站开发中，往往会遇到需要对用户进行区分的情况，不同的用户也往往拥有不同的权限或功能。此时，通过条件渲染，可以实现对于不同的用户种类渲染出不同页面的效果。

在例 3-14 中，实现了根据用户作为学生、老师、助教不同身份进行不同前端页面显示的效果。代码的运行结果如图 3-10 所示。

【例 3-14】 根据用户身份显示不同前端页面

```
<template>
  <div id="app">
    <h1>User Type:</h1>
    <template v-if="user === 'student'">
      <h2>{{ user }}</h2>
      <br />
      <button @click="joinClass">加入课程</button>
    </template>
    <template v-if="user === 'teacher'">
      <h2>{{ user }}</h2>
      <br />
      <button @click="startClass">开始上课</button>
    </template>
    <template v-if="user === 'teaching assistant'">
      <h2>{{ user }}</h2>
      <br />
      <button @click="manageStudent">管理学生</button>
    </template>
  </div>
</template>

<script>
import { reactive, toRefs } from "vue";
export default {
  setup() {
    const data = reactive({
      user: "teacher",
    });

    const joinClass = () => {
      console.log("Request already sent!");
    };

    const startClass = () => {
      console.log("Class begin!");
    };

    const manageStudent = () => {
      console.log("Checking requests!");
    };

    return {
      ...toRefs(data),
      joinClass,
      startClass,
      manageStudent,
    };
  },
};
</script>
```

(a) 教师身份　(b) 学生身份　(c) 助教身份

图 3-10　不同用户身份下的显示与输出

通过修改按钮的方法，可以实现对不同用户进行功能区分的效果。

3.6　v-for 列表渲染

3.6.1　v-for 指令

当需要将一个数组遍历或枚举一个对象循环显示时，就会用到列表渲染指令 v-for 来渲染一个列表。它的表达式需结合 in 来使用，类似 item in items 的形式。其中，items 是源数据数组，而 item 则是被迭代的数组元素的别名，也可以用 of 替代 in 作为分隔符。

v-for 指令也可以像 v-if 一样用在内置标签< template >上渲染多个元素。在自定义组件上同样可以使用 v-for 指令。但是由于组件有自己独立的作用域，数据不会被自动传递到组件中。此时，需要使用 props 把迭代数据传递到组件里。

在 v-for 块中，我们可以访问所有父作用域的属性。v-for 指令还支持一个可选的第二个参数 index，即当前项的索引，索引从 0 开始计数。具体代码如例 3-15 所示，其实现效果如图 3-11 所示。

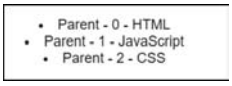


图 3-11　列表渲染

【例 3-15】 用 v-for 指令显示数组中的 message 元素

```
< template >
  < div id = "app">
    < li v - for = "(item, index) in items">
     {{ parentMessage }} - {{ index }} - {{ item.message }}
    </li >
  </div >
</template >

< script >
import { reactive, toRefs } from "vue";
export default {
  setup() {
    const data = reactive({
      parentMessage: "Parent",
      items: [
        { message: "HTML" },
        { message: "JavaScript" },
        { message: "CSS" },
```

```
      ],
    });

    return {
      ...toRefs(data),
    };
  },
};
</script>
```

3.6.2 在 v-for 里使用对象

使用 v-for 可以遍历一个对象的属性；也可以设置第二个参数为属性的名称，即键名 key；还可以用第三个参数作为索引 index。通过 v-for 遍历对象的代码如例 3-16 所示。对象中存储了一些书籍的属性，通过 index、name 和 value 分别获取属性的索引、名称和值，并在视图中显示。遍历对象属性的结果如图 3-12 所示。

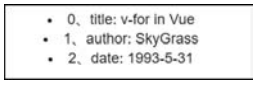


图 3-12　遍历对象属性结果

【例 3-16】 用 v-for 遍历对象属性

```
<template>
  <div id="app">
    <li v-for="(value, name, index) in obj">
      {{ index }}、{{ name }}: {{ value }}
    </li>
  </div>
</template>

<script>
import { reactive, toRefs } from "vue";
export default {
  setup() {
    const data = reactive({
      obj: {
        title: "v-for in Vue",
        author: "SkyGrass",
        date: "1993-5-31",
      },
    });

    return {
      ...toRefs(data),
    };
  },
};
</script>
```

3.6.3 列表的更新

Vue 的核心是数据与视图的双向绑定，当修改数组时，Vue 会检测到数据变化，所以用 v-for 指令渲染的视图也会立即更新。Vue 将被侦听数组的变更方法进行了包裹，所以使用它们改变数组会触发视图更新。数组的变更方法如表 3-2 所示。

表 3-2 数组的变更方法

名　称	说　明
push()	将一个或多个元素添加至数组末尾，返回新数组的长度
pop()	从数组中删除并返回最后一个元素
shift()	从数组中删除并返回第一个元素
unshift()	将一个或多个元素添加至数组开头，并返回新数组的长度
splice()	从数组中删除元素或向数组添加元素
sort()	对数组元素排序，默认按照 Unicode 编码排序，返回排序后的数组
reverse()	将数组中的元素位置颠倒，返回颠倒后的数组

使用表 3-2 所示的方法会改变被这些方法调用的原始数组，此外还有一些方法不会改变原数组，如 filter()、concat()、slice()，它们返回的是一个新数组。

需要注意的是，当直接使用下标或键名为数组或对象设置成员，以及修改数组长度时，Vue 并不会将其加入数据响应式系统，如 arr['index']='value'、obj['key']='value'或 arr.length=1。此时即使数据被修改，视图也不会进行更新。

3.6.4 列表渲染的 key

在使用 v-for 指令时，最好为每个迭代元素提供一个值不重复的 key，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。因为，当列表渲染被重新执行(数组内容发生改变)时，如果不使用 key，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被正确渲染。

通过为每项提供一个唯一的 key，给 Vue 进行提示，Vue 就会根据 key 的变化重新排列节点顺序，并移除 key 不存在的节点。实质上，key 的存在是为 DOM 节点标注了一个身份信息，让 Vue 能够有迹可循地追踪到数据对应的节点。

在实战开发中，是否使用 key 都不会影响功能的实现。建议尽可能在使用 v-for 指令时提供 key，如例 3-17 所示。

【例 3-17】 指定 v-for 的 key

```
<div v-for="item in items" :key="item.id">
  <!-- content -->
</div>
```

3.6.5 v-for 与 v-if 指令共用

在多数情况下，不推荐在同一元素上同时使用 v-if 和 v-for 指令。但当它们处于同一节

点时，v-if 指令的优先级比 v-for 指令更高，因此 v-if 指令将没有权限访问 v-for 指令里的变量，可以通过把 v-for 指令移动到外层的< template >标签中来实现相同的效果。

3.6.6 列表渲染实例：帖子列表

学习完列表渲染 v-for 与条件渲染 v-if，可以尝试制作一个帖子列表。帖子带有收藏功能，被收藏的帖子标题将会以红色显示，并带有收藏与取消收藏按钮。

具体代码如例 3-18 所示，代码的运行结果如图 3-13 所示。通过修改 addStar(item)方法的内容，可以实现不同的收藏状态。在此基础上，也可以引申出置顶、删除帖子等许多方式。

【例 3-18】 帖子列表

```
<template>
  <div id="app">
    <div v-for="(item, index) in posts" :key="index" class="post">
      <h1 v-if="item.isStar === true" style="color: red">
        {{ item.title }}
      </h1>
      <h1 v-if="item.isStar === false">
        {{ item.title }}
      </h1>
      <div class="content">
        {{ item.detail }}
      </div>
      <button class="star" v-if="item.isStar === false" @click="addStar(item)">
        收藏
      </button>
      <button class="star" v-if="item.isStar === true" @click="addStar(item)">
        取消收藏
      </button>
    </div>
  </div>
</template>

<script>
import { reactive, toRefs } from "vue";
export default {
  setup() {
    const data = reactive({
      posts: [
        { title: "HTML", detail: "Something about HTML.", isStar: false },
        {
          title: "JavaScript",
          detail: "Something about JavaScript.",
          isStar: true,
        },
```

```
            { title: "CSS", detail: "Something about CSS.", isStar: false },
          ],
        });

        const addStar = (item) => {
          //do something
        };

        return {
          ...toRefs(data),
          addStar,
        };
      },
    };
    </script>
```

图 3-13　帖子列表的显示

3.7　v-on 事件绑定

3.7.1　v-on 指令

v-on 指令用于监听 DOM 事件，并在触发事件时执行一些 JavaScript，通常缩写为@符号。直接把 JavaScript 代码写在 v-on 指令中是一种方式。但是许多事件的处理过程十分复杂，因此 v-on 还可以接收一个需要调用的方法名称。用法为 v-on:click="methodName" 或使用快捷方式@click="methodName"。

不难发现，事件绑定已经在之前的例子中出现过很多次。按钮的单击事件@click 即等同于 v-on:click。除了单击事件，v-on 指令后面还可以接其他 HTML 的标准事件，如表 3-3 所示。

表 3-3　部分 HTML 的标准事件

事　　件	说　　明
click	单击
dblclick	双击
contextmenu	右击
mouseover	指针移到有事件监听的元素或其子元素内
mouseout	指针移出元素，或者移到其子元素上
keydown	键盘动作：按下任意键
keyup	键盘动作：释放任意键

3.7.2　事件修饰符

Vue 为 v-on 指令提供了事件修饰符，修饰符是由点开头的指令后缀来表示的。使用多个修饰符时，顺序很重要。常见的事件修饰符见表 3-4。

表 3-4　常见的事件修饰符

名　　称	可 用 事 件	说　　明
. stop	任意	当事件触发时，阻止事件冒泡
. prevent	任意	当事件触发时，阻止元素默认行为
. capture	任意	当事件触发时，阻止事件捕获
. self 自身	任意	限制事件仅作用于节点自身
. once	任意	事件被触发一次后即解除监听
. passlve	滚动	移动端，限制事件永不调用 preventDefault()方法

Vue 允许为 v-on 或者@在监听键盘事件时添加按键修饰符，用于检查详细的按键。使用方法为@keyup. enter="methodName"。以下是全部的别名：

(1) . enter；

(2) . tab；

(3) . delete(“删除”和“退格”)；

(4) . esc；

(5) . space；

(6) . up；

(7) . down；

(8) . left；

(9) . right。

除了键盘按键之外，Vue 也为鼠标按键配置了修饰符，包括. left、. right 与. middle。

当需要按住多键或者鼠标与键盘共用以实现操作时，可以使用组合修饰符，需要配合系统按键修饰使用，包括. ctrl、. alt、. shift 与. meta。在 macOS 系统键盘上，meta 对应 command 键；在 Windows 系统键盘上，meta 对应 Window 键。

3.8 指令在 Vue 3.x 中的变化

3.8.1 v-if 与 v-for 的 key

v-if/v-else/v-else-if 的各分支项 key 属性不再是必需的，因为即使没有为条件分支提供 key，Vue 3.x 也会自动生成唯一的 key。因此，不建议在 v-if/v-else/v-else-if 的分支中继续使用 key 属性。如果主动提供 key，那么每个分支必须使用唯一的 key。故意使用相同的 key 也无法强制重用分支。

在 Vue 3.x 中，<template v-for>的 key 应该设置在<template>标签上，而不是设置在它的子节点上。在 Vue 2.x 中，由于<template>标签不能拥有 key，可以为其每个子节点分别设置 key。同样，当使用<template v-for>时存在使用 v-if 的子节点，key 应改为设置在<template>标签上。

3.8.2 v-if 与 v-for 的优先级

在 Vue 2.x 版本中，在一个元素上同时使用 v-if 和 v-for 时，v-for 会优先作用。在 Vue 3.x 版本中，v-if 拥有比 v-for 更高的优先级。

由于语法上存在歧义，建议避免在同一元素上同时使用两者。具体可以参考 3.6.1 节中提到的方法。

3.8.3 v-bind 合并行为

在元素上动态绑定 attribute 时，在一个元素中同时使用 v-bind="object"语法和单独的 property 是一种常用的做法。这就涉及了合并的优先级的问题。

在 Vue 2.x 中，如果一个元素同时定义了 v-bind="object"和一个相同的单独的 property，那么这个单独的 property 总是会覆盖 object 中的绑定。在 Vue 3.x 中，声明绑定的顺序决定了它们如何合并，开发者对自己所希望的合并行为有了更好的控制。

如果希望依赖 v-bind 的覆盖功能，建议确保在单独的 property 之前定义 v-bind 属性。

3.8.4 v-for 中的 ref 数组

在 Vue 2.x 中，v-for 里使用的 ref attribute 会用 ref 数组填充相应的 $refs property。但是，当存在嵌套的 v-for 时，会变得不明确且效率低下。

在 Vue 3.0 中，将不再在 $ref 中自动创建数组。要从单个绑定获取多个 ref，可以将 ref 绑定到一个更灵活的函数上，如例 3-19 所示。

【例 3-19】 ref 绑定到函数

```
<template>
  <div id="app">
      <div v-for="item in list" :ref="setItemRef"></div>
  </div>
</template>
```

```
<script>
import { ref, onBeforeUpdate, onUpdated } from 'vue'

export default {
  setup() {
    let itemRefs = []
    const setItemRef = el => {
      itemRefs.push(el)
    }
    onBeforeUpdate(() => {
      itemRefs = []
    })
    onUpdated(() => {
      console.log(itemRefs)
    })
    return {
      itemRefs,
      setItemRef
    }
  }
}
</script>
```

值得注意的是，itemRefs 不必是数组，它也可以是一个对象，其 ref 会通过迭代的 key 被设置。itemRefs 也可以是响应式的且被监听。

3.8.5 v-model

在用于自定义组件时，v-model 的 prop 和事件默认名称已更改。prop 从 value 变更为 modelValue，event 从 inpu 变更为 update：modelValue。v-bind 的.sync 修饰符和组件的 model 选项已移除，用 v-model 作为代替。可以在同一个组件上使用多个 v-model 进行双向绑定。

除了像.trim 这样的 Vue 2.x 硬编码的 v-model 修饰符外，Vue 3.0 还支持自定义 v-model 修饰符。

迁移时，可以将所有使用.sync 的部分替换为 v-model。对于所有不带参数的 v-model，确保分别将 prop 和 event 命名更改为 modelValue 和 update:modelValue。

更多关于 v-model 与组件的更新将会在第 7 章进行说明。

3.9 本章小结

本章介绍了插值绑定、计算属性、属性绑定、双向绑定、条件渲染、列表渲染、事件绑定以及 Vue 的内置指令从 Vue 2.x 到 Vue 3.x 的变化。

文本插值最基本的方法是使用双大括号(Mustache 语法)“{{ }}”，Vue 会将大括号里的内容替换为表达式值，以文本的形式将其展示出来。表达式可由 JavaScript 表达式和过滤器构成。通过任何方法修改数据设定值，大括号的内容都会被实时替换。Vue 支持在

“{{ }}”插值的尾部添加过滤器，用管道符“|”表示。

计算属性会在其依赖属性的值发生变化时，对属性的值进行自动更新，同时更新相关的DOM部分。通过从Vue中导入computed来使用计算属性。侦听属性watch用来监视指定数据项的变化，从而触发用户自定的操作。默认情况下，回调仅在侦听的数据源发生改变时调用。通过从Vue中导入watch来使用侦听属性。

v-bind指令主要用于动态绑定DOM元素属性，可以将一个或多个attribute，或一个组件prop动态地绑定到表达式。

v-model指令用于在< input >、< textarea >及< select >等表单控件元素上创建双向绑定，它会根据控件类型自动选取正确的方法来更新元素。在修改表单元素值时，对应的实例中的属性值会被同时更新。

Vue根据表达式的值，在DOM中渲染或销毁元素与组件，称为条件渲染。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真值的时候被渲染。当v-if中的表达式返回值为假时，根据v-else-if表达式值的真假进行渲染。当v-else-if与v-if的表达式值均为假时，渲染v-else中的内容。v-show的用法与v-if大致相同。因为v-show只是切换元素CSS属性的display，所以v-show的元素始终会被渲染并保留在DOM中。

使用列表渲染指令v-for可以渲染一个列表，用于遍历一个数组或枚举一个对象循环显示。它的表达式需结合in来使用。

v-on指令用于监听DOM事件，并在触发事件时执行一些JavaScript，通常缩写为@符号。

在指令方面，Vue 3.0的更新集中在v-if与v-for的key、v-if与v-for的优先级、v-bind合并行为、v-for中的ref数组和v-model。

3.10 练 习 题

一、填空题

1. v-show与v-if的区别在于，无论条件真与否，_____都会被编译。

2. 可以在Vue中导入_____使用侦听属性。

3. 在使用v-for时，最好为每个迭代元素提供一个值不重复的_____，以便它能跟踪每个节点的身份。

4. 在展示数据时，往往需要在不变更或重置原始数据的情况下，显示经过过滤或排序的数组。在这种情况下，可以创建一个_____属性返回过滤或排序后的数组。

5. 在Vue 3.x版本中，v-if和v-for优先级更高的是_____。

二、单选题

1. Vue中格式化文本时应使用(　　)。

A. 文本插值　　B. 过滤器　　C. HTML插值　　D. 计算属性

2. v-model的修饰符中可以删除用户输入的首尾空白字符的是(　　)。

A. .lazy　　B. .number　　C. .tmn　　D. length

3. HTML的标准事件中，代表“按下任意键”的是(　　)。

A. mouseover　　B. keydown　　C. keyup　　D. click

4. 对于值是HTML的片段，可以使用的绑定方法是(　　)。

A. {{{}}}　　B. {{}}　　C. {}　　D. []

5. 列表更新中从数组中删除元素或向数组添加元素的方法是(　　)。

A. push()　　B. shift()　　C. splice()　　D. pop()

三、判断题

1. 在进行相等的条件判断时,应该使用==。(　　)

2. 使用 v-for 可以遍历一个对象的属性,可以用第三个参数作为索引 key。(　　)

3. 直接使用下标或键名为数组或对象设置成员,以及修改数组长度时,即使数据被修改,视图也不会进行更新。(　　)

4. 当处于同一节点时,v-if 的优先级比 v-for 更高。(　　)

5. 当需要绑定一个动态的数据时,可以用 v-model 实现。(　　)

四、问答题

1. v-if 与 v-for 在同一元素上使用时优先级是怎样的?如何避免这种情况?

2. 在文本插值中,如何实现简单的条件判断?

3. 简要说明如何创建仅可读和可读可写的计算属性。

4. 分别说明几种修饰符如何控制 v-model 中数据同步的时机。

5. 列举 v-if 与 v-show 的异同。

6. 解释 v-for 中各参数以及 key 的作用。

7. 列举什么情况下对 v-for 数组与对象的更新不会导致视图的更新。

五、动手做

尝试使用 v-model、v-if 与 v-for 的知识,制作一个简易贴吧。有普通用户和管理员两种角色,普通用户可以发帖、收藏帖子;管理员可以发帖、置顶帖子。