第3章


视频讲解

Vue数据绑定

本章学习目标

- 理解和掌握 Vue 中数据绑定原理。
- 理解单向和双向数据绑定工作过程和原理。
- 掌握绑定文本和指令绑定数据的方法。

3.1 Vue 中数据绑定原理

Vue 中最大的一个特征就是数据的双向绑定，而这种双向绑定的形式，一方面表现在元数据与衍生数据之间的响应，另一方面表现在元数据与视图之间的响应，而这些响应的实现方式，依赖的是数据链，因此，要了解数据绑定的原理，先要理解下面两方面内容。

3.1.1 Vue 中的数据链

数据链是一种数据关联的形式，在这种形式中，一或多个起始数据点，称为元数据，而由这些元数据因某种关系衍生出的数据，称为衍生数据，元数据与衍生数据通过数据结点交织在一起，形成数据结构网，而这种结构网，称为数据链，如图 3-1 所示。

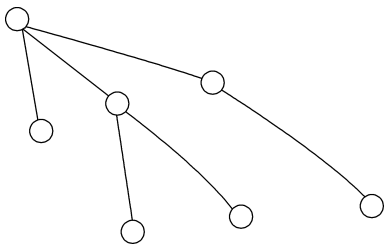
图 3-1 数据链模式

在 Vue 中，当数据链中的元数据变化时，与其关联的衍生数据，通过数据链完成同步更新，实现数据双向绑定的效果。在 Vue 实例化对象中，computed 选项值可以为开发人员生成衍生对象，当元数据变化时，生成的衍生对象将会同步更新。

实例 3-1 使用衍生数据显示“张三，你好！”

1. 功能描述

新建一个名称为 SayHello 的 vue 组件，在返回的数据对象中，添加一项名称为“name”的属性，初始值为“张三”。同时，在 computed 配置选项中，添加一个名为“sayHelloName”的函数，在函数中返回“张三，你好！”，并在页面中执行该函数。

2. 实现代码

在项目的 components 文件夹中，新建一个名称为“ch3”的子文件夹，在这个子文件夹中添加一个名为“SayHello”的. vue 文件，在文件中加入如代码清单 3-1 所示的代码。

代码清单 3-1　SayHello. vue 代码

```
<template>
  <div>
    <div>{{ name }}</div>
    <div>{{ sayHelloName }}</div>
  </div>
</template>
<script>
export default {
  data() {
    return {
      name: "李四",
    };
  },
  computed: {
    sayHelloName() {
      return this.name + ",你好!";
    }
  },
};
</script>
<style scoped>
div {
  margin: 10px;
  text-align: left;
}
</style>
```

SayHello 文件是一个独立的 vue 组件，需要将它导入根组件 App. vue 中，并声明该组件，最后在模板中以标签形式使用该组件。因此，App. vue 文件修改后的代码，如代码清单 3-2 所示。

代码清单 3-2　App. vue 代码

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

<script>
import SayHello from "./components/ch3/SayHello.vue";

export default {
  name: "App",
  components: {
    SayHello
  }
};
```

```
</script>

<style>
...省略样式代码
</style>
```

3. 页面效果

保存代码后,页面在 Chrome 浏览器下执行的效果如图 3-2 所示。

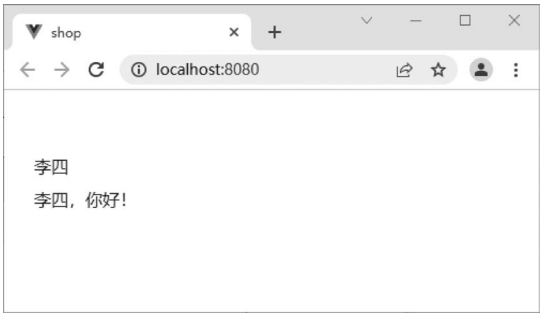

图 3-2 使用数据链输出内容

4. 源码分析

在 Vue 实例化配置对象中,computed 选项可以生成衍生数据,生成过程由函数来完成,该函数不接收参数,在函数体中,由于 this 指向实例化的 Vue 对象,因此,它可以访问所有实例化对象中挂载的属性值,如 this.name,表示元数据值。

此外,computed 选项中的函数,虽然可以访问实例化对象中挂载的全部属性值,但它在函数中必须使用 return 语句,返回计算或衍生后的数据,通过这种形式,才可以在模板中使用双大括号方式执行函数,接收并显示返回的数据。

因此,示例中的 sayHelloName()函数中,先获取元数据 name 值,并添加“,你好!”形成一个衍生数据,并作为函数的返回值; 当在模板中调用该函数时,则直接将接收到的衍生数据显示在页面中,由于是衍生数据,当元数变化时,将会通过数据链形式同步衍生数据,因此,衍生数据也会同步更新。即修改 name 值为“李四”时,页面将直接显示“李四,你好!”的字样。

3.1.2 数据绑定视图

一般而言,一个对象是由多个 key/value 键值对组成的无序集合,并且对象中的每个属性值可以是任意类型的,向对象添加属性值时,可以是字面量或构建函数,如以下代码。

```
var obj = new Object;          //等价于 obj = {}
obj.name = "张三";             //添加描述
obj.say = function(){};        //添加行为
```

除上述方式之外,还可以使用 Object.defineProperty()方法定义新属性或修改原有的属性值。在设置和获取属性时,可以使用 setter 和 getter 方法,前者用于设置对象的属性值,后者用于获取对象的属性值,如以下代码。

```
var obj = {};
var initValue = 'hello';
Object.defineProperty(obj, "name2", {
     get: function () {
           //函数在获取值时触发
           return initValue;
     },
     set: function (value) {
           //函数在设置值时触发,新值通过参数 value 获取
           initValue = value;
     }
});
//获取初始值
console.log(obj.name2);          //输出"hello"
//设置新值
obj.name2 = '李四';
//输出设置的值
console.log(obj.name2);          //输出"李四"
```

对象中的 setter 和 getter 方法不需要成对出现，根据需求可以单独添加。由于它们都是方法，因此，可以在执行方法过程中执行其他的功能，例如，在修改属性值时，重置页面中显示属性值的元素内容，实现数据同步绑定视图内容的效果，实例如下。

实例 3-2　数据同步绑定视图内容

1. 功能描述

在新建的页面中添加一个文本输入框元素，并添加一个 div 元素，当在文本框中输入内容时，div 元素中同步显示文本框中输入的内容。

2. 实现代码

在项目 components 文件夹的 ch3 子文件夹中，添加一个名为“dataView”的.html 文件，在文件中加入如代码清单 3-3 所示的代码。

代码清单 3-3　dataView.html 代码

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

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>同步视图绑定的数据</title>
</head>

<body>
    <input type="text" id="txt">
    <div id="tip">...</div>
    <script>
        let txt = document.getElementById("txt");
        let tip = document.getElementById("tip");
```

```
        let obj = {
            name: ""
        }
        let temp = {}
        let propName = "name";
        Object.defineProperty(obj, propName, {
            set(v) {
                tip.innerHTML = v;
                temp[propName] = v;
            },
            get() {
                return temp[propName];
            }
        })
        txt.addEventListener("keyup", function () {
            obj[propName] = this.value;
        })
    </script>
</body>

</html>
```

3. 页面效果

保存代码后,页面在 Chrome 浏览器下执行的效果如图 3-3 所示。

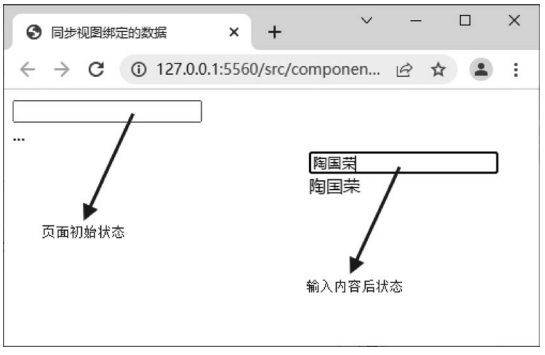

图 3-3 数据同步绑定视图内容

4. 源码分析

在本实例的页面源码中,首先使用对象的 defineProperty()方法获取和重置 obj 对象的 name 属性值;其次,在重置方法中,不仅修改了对象属性值,还将该属性值赋给 div 元素;最后,当文本框执行绑定的 keyup 事件时,需要重置对象的 name 属性,因此,触发了对象属性变更的 setter 函数,在重置属性值时,将属性值同步显示在元素的内容中,这个过程实质上也是 Vue 中数据与视图绑定的原理。

在 Vue 中,当把 JavaScript 对象传给 Vue 实例的 data 选项时,Vue 将遍历这个对象的全部属性,并使用 Object. defineProperty 将其全部转换为 getter/setter 变更形式,并在组件渲染时,记录属性与视图的依赖关系。根据这种依赖关系,当 setter 函数再次被调用时,会通知 watcher 重新计算并更新其关联的所有组件,最终实现数据双向同步的功能。

3.2　单向数据绑定

Vue是一个典型的MVVM框架，那么，什么是MVVM框架？它与MVC框架相比，有了哪些改变？单向数据绑定在这个框架中是如何体现的？带着这些问题，进入接下来的内容学习。

3.2.1　MVC框架演变过程

严格来说，MVC框架是一种设计思想，早期的前端技术MVC结构来源于后端语言，如Java、C#语言，这些语言具有完整和成熟的MVC框架体系。随着前端处理业务的逻辑越来越复杂，便借鉴后端语言这种MVC框架体系，形成前端技术特有的MVC框架。

它的结构与后端语言的MVC一样，由Model、View、Controller三部分组成，它们各司其职，Model简称M，即数据模型层，用于定义数据结构和存储数据源；View简称V，即视图层，用于展示数据界面和响应页面交互；Controller简称C，即控制层，用于监听数据变化并处理页面交互逻辑，它们三者的关系如图3-4所示。

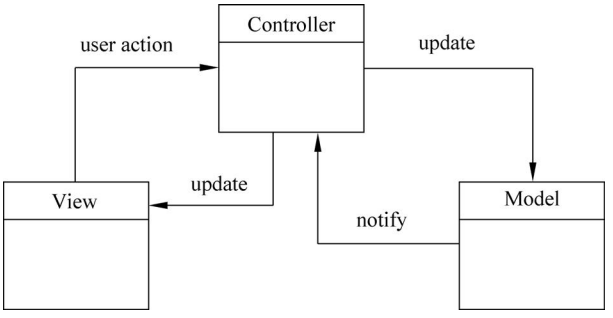

图3-4　MVC数据流向示意图

但随着业务逻辑越来越复杂，Controller层代码量也越来越多，显得冗余而无序，维护起来非常困难。这时，需要从Controller层抽离出数据和逻辑处理部分，由专门的一个对象进行管理和维护，而这个对象，就是ViewModel。

通过抽离出ViewModel对象，逻辑层的结构更加清晰，ViewModel负责处理视图和数据逻辑关系，并双向绑定View和Model，使ViewModel对象更像一座桥梁，用于衔接View层和Model层两端，它们的关系如图3-5所示。

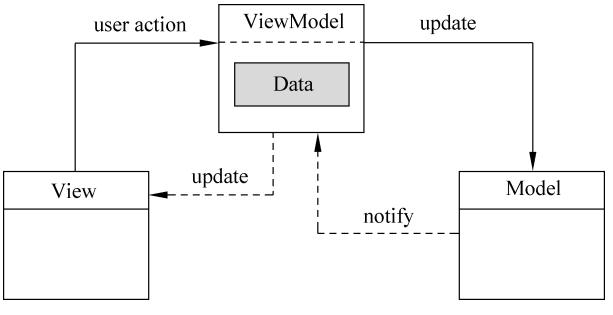

图3-5　MVVM数据流向示意图

在图 3-5 中不难看出,原来 Controller 层需要处理所有的数据交互和业务逻辑,而改成 ViewModel 层后,只需要处理针对 View 层的数据交互和业务逻辑,并且这种处理后的绑定是双向的,这样就使 View 和 Model 层的数据同步是完全自动的,用户无须手动操作 DOM,只须关注业务逻辑。

3.2.2 单向绑定

在 MVVM 框架下,Vue 的数据绑定都是双向的,但也能实现单向的数据绑定。所谓"单向"是针对"双向"而言的,也就是一个方向,即从数据源获取数据,到视图层中显示数据一个方向,在显示时并不会改变源数据,这种单向绑定的方式常用于绑定视图层中元素固定显示的内容、元素属性中,实例如下。

实例 3-3 数据单向绑定

1. 功能描述

在新建的组件中,添加一个 div 和 span 元素,并使用单向数据绑定的方式,显示 span 元素的内容和控制元素的类别样式。

2. 实现代码

在项目 components 文件夹的 ch3 子文件夹中,添加一个名为"OneWay"的. vue 文件,在文件中加入如代码清单 3-4 所示的代码。

代码清单 3-4 OneWay. vue 代码

```
<template>
  <div>
    <div>
      姓名:<span v-bind:class="fs">{{ name }}</span>
    </div>
  </div>
</template>
<script>
export default {
  data() {
    return {
      name: "李小明",
      fs: "fs",
    };
  },
};
</script>
<style scoped>
.fs {
  font-size: 26px;
  color: red;
}
div {
  margin: 10px;
  text-align: left;
```

```
}
</style>
```

3. 页面效果

保存代码后，页面在 Chrome 浏览器下执行的效果如图 3-6 所示。

图 3-6　数据单向绑定

4. 源码分析

在组件的实例化的配置对象中，先分别定义了“name”和“fs”对象属性，作为视图层绑定的数据源，然后在视图中通过 v-bind 指令绑定元素的 class 属性值，使用双大括号绑定元素显示的内容，这种绑定的方式就是单向的数据绑定。最后，当数据源发生变更后，视图层将自动同步变更后的数据。

3.3　双向数据绑定

在 Vue 的 MVVM 框架下，既可以实现单向数据绑定，也能完成数据的双向绑定。所谓的“双向绑定”，可以简单理解为数据源变化后，绑定的视图发生相应变化；绑定视图的数据变化后，数据源也会发生同步的变化，这就是“双向绑定”。

简单的理解只是说明了它的数据状态，还需要进一步了解它背后的工作原理。要实现数据的双向绑定，需要添加三个核心的对象，分别是 Observer、Watcher 和 Complie。Observer(观察者)对象，用于收集所有需要绑定的对象属性。Watcher(订阅者)对象，用于接收对象属性的变化并处理变化后的逻辑。Complie(指令解析器)对象，用于初始化元素与对象属性数据的结构并构建属性变化后的逻辑。

在 Observer 和 Watcher 对象进行关联时，由于 Watcher 对象是订阅者，当 Observer 对象属性有变化时，就会自动告诉 Watcher 对象是否要同步更新，但因为类似于 Watcher 对象的订阅者有多个，便添加了一个 Dep(消息订阅器)来进行集中管理，由 Dep 来批量向 Watcher 对象传递是否要同步更新的指令。最后，它们之间的结构如图 3-7 所示。

在上述示意图中，当对象属性在视图中发生变化时，将通过 Dep 通知 Watcher 对象，Watcher 对象将根据 Complie 构建的规则，执行更新函数，最后将数据同步更新到视图元素中，从而实现数据双向绑定的效果。

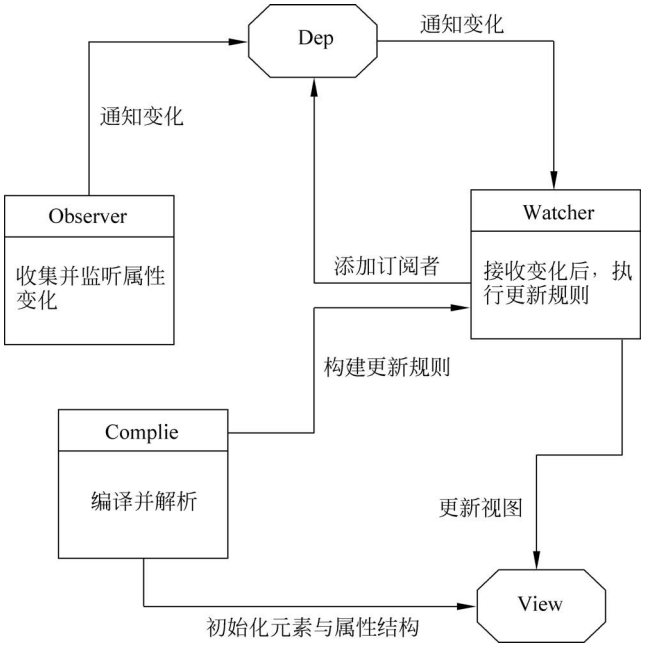

图 3-7　数据双向绑定工作原理示意图

3.3.1　指令 v-model

在 Vue 中，v-model 指令常用于表单的各元素中，它可以实现数据的双向绑定效果，即指令中元素的值绑定数据源，数据源变化后，元素的值也会跟随变化；同时，元素的值发生变化，绑定的数据源也会同步变化的值，实现双向同步数据的效果。

虽然这个指令有这么强大的功能，但它本质上是一个“语法糖”，双向绑定数据的背后，实际上是元素值和 oninput 事件共同被绑定后的结果，下面通过一个实例来说明。

实例 3-4　数据双向绑定

1. 功能描述

在实例 3-3 基础之上，再添加两个 input 元素，第一个直接使用 v-mode 指令绑定属性 name 的值，第二个先使用 v-bind 指令绑定元素的值，再绑定元素的 input 事件，在事件中，将获取的输入值赋值给 name 属性。

2. 实现代码

在项目 components 文件夹的 ch3 子文件夹中，添加一个名为“TwoWay”的. vue 文件，在文件中加入如代码清单 3-5 所示的代码。

代码清单 3-5　TwoWay. vue 代码

```
<template>
  <div>
    <div>指令绑定输入：<input type="text"
        v-model="name" /></div>
    <div>
```

```
        事件绑定输入：< input
          type = "text"
          v - bind:value = "name"
          @input = "name =  $event.target.value"
        />
      </div >
      < div >
        姓名：< span v - bind:class = "fs">{{ name }}</span >
      </div >
    </div >
</template >
< script >
//代码与实例 3 - 3 相同
</script >
< style scoped >
//样式与实例 3 - 3 相同
</style >
```

3. 页面效果

保存代码后，页面在 Chrome 浏览器下执行的效果如图 3-8 所示。

图 3-8　数据双向绑定

4. 源码分析

在上述实例的模板代码中，首先使用 v-model 方式将元素值与属性值进行绑定，当元素值发生变化后，被绑定的属性值同步变化。然后，使用 v-bind 方式将元素值与属性值绑定，再在元素的 input 事件中，将变化后的元素值再赋值给属性值。最后发现这两种方式的效果是一致的，说明它们的功能是相同的。

由此而见，v-model 指令方式实现数据的双向绑定，也依赖于监控，只不过这种监控并不是事件，而是 Watcher 对象，当被绑定订阅者的属性发生变化时，Watcher 对象就会获知，并根据 Complie 制定的更新规则，将源数据同步更新为变化后的属性值，并同时更新视图层，最终实现数据的双向绑定效果。

3.3.2　v-model 与修饰符

当表单中的元素与 v-model 指令绑定时，还可以通过“.”语法的方式添加修饰符，如

lazy、number 和 trim。lazy 用于延迟元素值与属性值更新的时机,input 事件中默认是同步更新,使用 lazy 修饰符后,数据更新的时机为 change 事件之后。

number 用于将更新的元素值转成数字型,这个修饰符非常有用,因为即使将元素的 type 类型设置为 number,获取的字符仍然是字符,因此,借助这个修饰符,可以将获取到的输入值快速转成 number 类型。

trim 用于删除元素值的首尾空格,使字符长度就是字符的内容。接下来通过一个实例来演示这三个修饰符的使用方法。

实例 3-5 model 与修饰符

1. 功能描述

新建一个组件,在模板元素中添加三个文本输入框并使用 v-model 指令绑定一个属性值。在这三个指令中,分别使用 lazy、number 和 trim 修饰符来描述 v-model 指令绑定的属性值,当元素值变化时,分别查看属性值的效果。

2. 实现代码

在项目 components 文件夹的 ch3 子文件夹中,添加一个名为“Modifier”的. vue 文件,在文件中加入如代码清单 3-6 所示的代码。

代码清单 3-6 Modifier. vue 代码

```
<template>
  <div>
    <div>
      lazy 修饰符: <input type="text"
     v-model.lazy="name" />{{ name }}</div>
    <div>
      number 修饰符: <input type="text"
     v-model.number="name" />{{ name + 1 }}
    </div>
    <div>
      trim 修饰符: <input type="text"
     v-model.trim="name" />{{ name.length }}
    </div>
  </div>
</template>
<script>
export default {
  data() {
    return {
      name: "123",
    };
  },
};
</script>
<style scoped>
div {
  margin: 10px;
  text-align: left;
}
```

```
input {
  padding: 8px;
  margin-right: 5px;
}
</style>
```

3. 页面效果

保存代码后，页面在 Chrome 浏览器下执行的效果如图 3-9 所示。

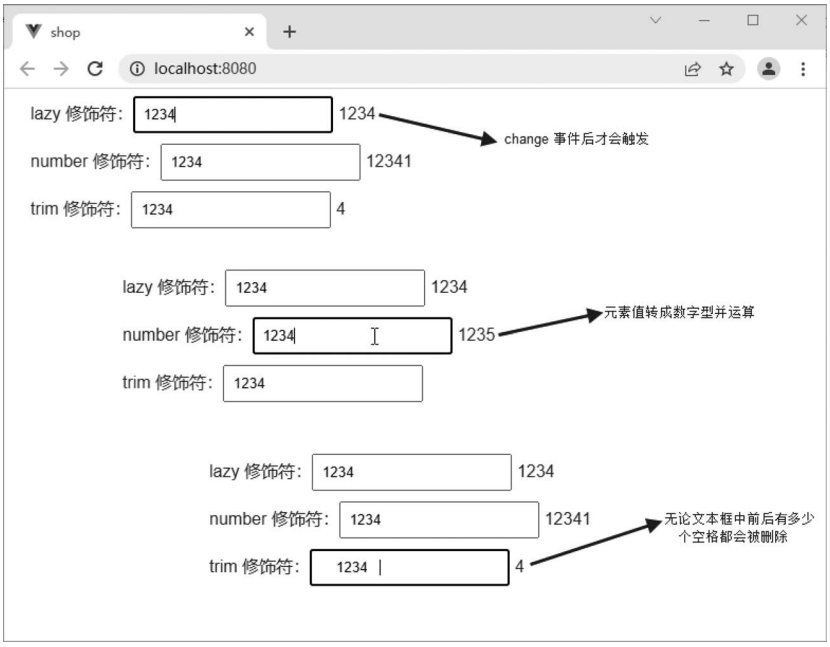

图 3-9　数据双向绑定

4. 源码分析

在本实例的模板中，当在第一行的文本框中输入一个数字 1234 时，由于添加了 lazy 修饰符，使文本框后的显示值，在 change 事件触发后才会同步更新。

当在第二行的文本框中输入一个数字 1234 时，由于添加了 number 修饰符，使输入框中的值自动转成数值，并与 1 相加，因此，文本框后的显示值为 1235。

当在第三行的文本框中输入一个数字 1234 时，虽然在文本框的前后位置都增加了空格，但由于使用了 trim 修饰符，使输入框中的值自动删除前后的空格，因此，文本框后的字符长度一直显示为 4。

3.4　数据绑定方法

在 Vue 中，数据绑定最常用的方法就是使用 Mustache 语法，这种语法的标签就是双大括号，因此又被称为双大括号插值语法。在这种语法下，双大括号标签将会被替换为绑定的属性值，并且，当属性值发生变化后，插值处的内容也会同步进行更新。

3.4.1 文本插值

文本插值是指使用 Mustache 语法绑定元素中显示的内容,如下。

```
<div>{{name}}</div>
```

使用这种方式插值后,如果 name 值发生了改变,那么,插值处元素的内容也会随之改变。当然,也可以不让它改变,只需要在这个元素上添加一个 v-once 指令,代码如下。

```
<div  v-once>{{name}}</div>
```

向元素添加了 v-once 指令后,元素中插入的值只是 name 属性的初始值,当该属性值变化后,插值处并不会随之改变,这种使用场景也有,但不是太多。

虽然 Mustache 语法可以向元素的内容插入数据,但它并不能使用这种方式向元素的属性插入数据,如果想绑定元素的属性,必须使用 v-bind 指令,并使用冒号":",指定绑定属性的名称,代码如下。

```
<div v-bind:class="red">{{name}}</div>
```

上述代码也等价于下列代码。

```
<div :class="red">{{name}}</div>
```

上述两行代码在浏览器中编译后,最终都为相同的一行代码,如下。

```
<div class="red" data-v-160690f0="">123</div>
```

需要说明的是,无论是 name 还是 red 属性,它们都是在 data()函数中定义好的对象属性名称,编译后,绑定的是这个属性名称对应的值。上述两行相同代码中,name 属性名对应的值是 123,red 属性名对应的值是"red",因此,才会显示上述编译后的相同代码。

3.4.2 JavaScript 表达式和 HTML 插值

Mustache 语法不仅可以向元素内容插入文本字符,同时,还可以在语法中插入简单的 JavaScript 表达式,如算术运算、三元运算和简单的函数调用,但只能执行单个表达式,不能执行语句。此外,通过向元素添加 v-html 指令,还可以向元素中插入 HTML 格式内容,接下来通过一个实例来详细说明它们的使用方法。

实例 3-6 JavaScript 表达式和 HTML 插值

1. 功能描述

新建一个组件,添加一个复选框和两个 div 元素,当选中复选框时,显示"插入 HTML 格式",同时,div 元素中内容以 HTML 格式显示; 当取消选中时,显示"插入文本格式",同时,div 元素中内容以文本格式显示。

2. 实现代码

在项目 components 文件夹的 ch3 子文件夹中，添加一个名为“BindHtml”的. vue 文件，在文件中加入如代码清单 3-7 所示的代码。

代码清单 3-7　BindHtml. vue 代码

```
<template>
  <div>
    <div>
      <input type="checkbox" v-model="blnHtml" />
      插入{{ blnHtml ? "HTML" : "文本" }}格式
    </div>
    <div v-show="!blnHtml">{{ HTML }}</div>
    <div v-show="blnHtml" v-html="HTML"></div>
  </div>
</template>
<script>
export default {
  data() {
    return {
      blnHtml: true,
      HTML: "<span style='font-weight:700'>你好,小明!</span>",
    };
  },
};
</script>
<style scoped>
div {
  margin: 10px;
  text-align: left;
}
</style>
```

3. 页面效果

保存代码后，页面在 Chrome 浏览器下执行的效果如图 3-10 所示。

图 3-10　JavaScript 表达式和 HTML 插值

4. 源码分析

在本实例的模板中，复选框使用 v-model 绑定的属性 blnHtml 是一个布尔值，既用于控制复选框元素的选中状态，又参与了不同内容显示的三元运算。同时，还作为 v-show 指

令绑定的属性值。该指令是一个用于控制元素是否显示和隐藏的指令,当该指令的属性值为 true 时,则元素显示,否则元素隐藏。

由于 v-html 指令用于控制元素的内容是否以 HTML 格式显示,且只能作用于元素的属性中,因此,通过再向元素添加一个 v-show 指令绑定 blnHtml 的属性值,实现 HTML 格式与文本格式切换显示的效果。

小结

本章先从数据链和数据绑定视图讲起,详细介绍了视图中数据绑定元素的工作原理,然后介绍单向和双向数据绑定的实现方式,并介绍修饰符的功能以及在绑定过程中添加修饰符的方法,最后详细说明如何在视图中插入文本和 HTML 格式的数据。