

详解 v-model


v-model 的基本使用

提到 Vue 中 v-model，同学们第一反应是用于在 input 框或者 textarea 标签上，用于进行数据绑定。

通常我们会这么使用：

```
<div id="app">
  <input type="text" v-model="msg">
  <p>{{ msg }}</p>
</div>

<script>
  new Vue({
    el: '#app',
    data: {
      msg: 'Hello Vue'
    }
  })
</script>
```



Hello Vue

在这里使用 v-model，实现了视图与数据之间的双向数据绑定。

当我们修改输入框中的内容，这里的 msg 也会发生变化，当我们手动修改 msg，input 当中的内容也会发生变化，这种形式也体现出双向数据绑定。

双向数据绑定，主要是由 MVVM 框架实现，在 Vue 中主要由三个部分组

成，View、ViewModel 和 Model 组成，其中 View 和 ViewModel 不能直接进行通信，他们要通过中间件 ViewModel 来进行。也就是当 Model 部分数据发生改变时，由于 vue 中 Data Binding 将底层数据和 Dom 层进行了绑定，ViewModel 通知 View 层更新视图；当在视图 View 数据发生变化也会同步到 Model 中。View 和 Model 之间的同步完全是自动的，不需要人手动操作 DOM。

接下来说 v-model 在 form 表单中的应用

v-model 与 form 表单

普通的输入框以及多行文本框的使用方式同上，主要目的实现数据与视图的同步。接下来从单选框，多选框，以及选项组来看看 v-model 的使用

单选按钮

单选按钮在单独使用时，可以直接使用 v-bind 绑定一个布尔类型的值，为真时选中，为假时不选。

```
<div id="app">
  <input type="radio" :checked="flag" >
  <p>{{ flag }}</p>
</div>

<script>
  let vm = new Vue({
    el: '#app',
    data: {
      flag: false
    }
  })
</script>
```

当我们的单选框是否被选中时我们可以通过 checked 属性进行绑定，具有 checked 属性的元素，说明被选中，不具有 checked 属性说明没有选

中。

如果想利用 Vue 实现单选框是否被选中，需要绑定点击事件，像这样。

```
<input type="radio" :checked="flag" @click="flag = !flag" >
```

这里的 flag 是布尔类型（true，false）

v-model 用在多个单选框构成的单选框组。

组合使用来实现互斥选择效果，就需要 v-model 配合 value 来使用：

```
<div class="app4">
  <input type="radio" id="html" value="html" v-model="picked" />
  <label for="html">HTML</label>
  <br />
  <input type="radio" id="js" value="js" v-model="picked" />
  <label for="js">JS</label>
  <br />
  <input type="radio" id="css" value="css" v-model="picked" />
  <label for="css">CSS</label>
  <br />
  <p>选择的项是： {{picked}}</p>
</div>
```

```
var app=new Vue({
  el:'.app4',
  data:{
    picked:'js'
  }
});
```

在这里我们使得单选框互斥，这个也就是 v-model 实现的功能，如果正常来说想实现互斥通过具有相同的 name 值。

复选框：

复选框也分单独使用和组合使用，不过用法稍与单选不同。复选框单独使用时，也是用 v-model 来绑定一个布尔值。也可像上面的方式进行绑定事件。

```
<input type="checkbox" v-model="flag" >
<p>{{ flag }}</p>
```

或者:

```
<input type="checkbox" :checked="flag" @click="flag = !flag" >
<p>{{ flag }}</p>
```

组合使用时，也是 `v-model` 与 `value` 一起，多个勾选框都绑定到同一个数组类型的数据，`value` 的值在数组当中，就会选中这一项。这个过程也是双向的，在勾选时，`value` 的值也会自动 `push` 到这个数组中：

```
<div class="app6">
  <input type="checkbox" name="" id="html" value="html" v-
model="checked" />
  <label for="html">HTML</label>
  <br />
  <input type="checkbox" name="" id="js" value="js" v-model="checked" />
  <label for="js">JS</label>
  <br />
  <input type="checkbox" name="" id="css" value="css" v-
model="checked" />
  <label for="css">CSS</label>
  <br />
  <p>选择的项是: {{checked}}</p>
</div>
```

```
var app=new Vue({
  el:'.app6',
  data:{
    checked:['html','css']
  }
});
```

选择列表局势下拉选择器，也是常见的表单控件，同样也分为单选和多选两种方式。`<option>`是备选项，如果含有 `value` 属性，`v-model` 就会优先匹配 `value` 的值；如果没有，就会直接匹配`<option>`的 `text`，比如选中第二项时，`selected` 的值是 `js` 而不是 `JS`。给`<selected>`添加属性 `multiple` 就可以多选了，此时 `v-model` 绑定的是一个数组，与复选框用法类似。

在业务中，`<option>`经常用 `v-for` 动态输出，`value` 和 `text` 也是用 `v-bind` 来动态输出的。

```
<div class="app9">
  <select name="" v-model="selected">
```

```

    <option
      v-for="option in options"
      :value="option.value"
      >{{option.text}}</option>
  </select>
  <p>选择的项是: {{selected}}</p>
</div>

```

```

var app=new Vue({
  el:'.app9',
  data:{
    selected:'html',
    options:[
      {
        text:'HTML',
        value:'html'
      },
      {
        text:'JavaScript',
        value:'js'
      },
      {
        text:'CSS',
        value:'css'
      }
    ]
  }
})

```

虽然用选择列表<select>控件可以很简单的完成下拉选择的需求，但是在实际业务中反而不常用，因为它的样式依赖平台和浏览器，无法统一，也不太美观，功能也有限，比如不支持搜索，所以常见的解决方案是用 div 模拟一个类似的控件。

v-model 语法糖的底层以及可配置

v-model 用在 input 元素上时

v-model 虽然很像使用了双向数据流，但是 Vue 是单项数据流，v-model 只是语法糖而已：

```
<input v-model="sth" />
<input v-bind:value="sth" v-on:input="sth = $event.target.value" />
```

第一行的代码其实只是第二行的语法糖。然后第二行代码还能简写成这样：

```
<input :value="sth" @input="sth = $event.target.value" />
```

要理解这行代码，首先你要知道 input 元素本身有个 oninput 事件，这是 HTML5 新增加的，类似 onchange，每当输入框内容发生变化，就会触发 oninput，把最新的 value 传递给 sth。

我们仔细观察语法糖和原始语法那两行代码，可以得出一个结论：

在给 <input /> 元素添加 v-model 属性时，默认会把 value 作为元素的属性，然后把 'input' 事件作为实时传递 value 的触发事件

v-model 用在组件上时

v-model 不仅仅能在 input 上用，在组件上也能使用。

组件的 v-model 生效原则：

接受一个 value 属性

在有新的 value 时触发 input 事件

父组件中在子组件上使用 v-model，默认会用 value 的 prop 来接受父组件 v-model 绑定的值，然后子组件通过 input 事件将更新后的值传递给父组件

child 组件中

```
<input :value="value"
      @input="onChildClick($event.target.value)">
props:{
```

```

    value: {
      type: String,
      default: ""
    }
  },
  methods: {
    onChildClick(value) {
      // 需要将更新后的值传递给父组件
      this.$emit('input', value)
    }
  }
}

```

、父组件中

```

// 相当于<child :value="name" @input="value => name = value"></child>
<child v-model="name"></child>
data(){
  return{
    name:''
  }
}

```

可配置的 v-model

对于 v-model 来说，我们前面说明 相当于

```
<input :value="val" @input="value =>val = value">
```

用在组件上也是相当于在组件上绑定 value 数据，绑定 input 的事件

在组件内部可以通过 model 字段进行配置

```

<div id="app">
  <my-inp v-model="value"></my-inp>
  <p>{{ value }}</p>
</div>

```

```

<script>
  // let obj = {
  //   flag: false
  // }

```

```
// }
let vm = new Vue({
  el: '#app',
  data: {
    flag: false,
    value: ''
  },
  components: {
    myInp: {
      //
      template: `<div>
        <input
          :value="val"
          @input="returnVal"/>
        </div>`,
      // props: ['value'] 默认配置 value
      model: {
        event: 'tell', //配置 v-model 的默认事件
        prop: 'val' //配置 v-model 的默认绑定值
      },
      props: ['val'],
      methods: {
        returnVal(e) {
          // 这里出发的是修改之后的默认事件，在这里是 tell 事件
          this.$emit('tell', e.target.value)
        }
      }
    }
  }
})
```

组件中通过 `model` 配置对象进行默认事件以及默认值的配置就可以了。