

06-v-model进阶+ref+nextTick

一、v-model进阶

1. 剖析原理

1.1 作用

实现双向数据绑定：

- 数据变了，视图跟着变
- 视图变了，数据跟着变

1.2 原理

v-model本质上是一个语法糖。

例如应用在输入框上，就是 value 属性 和 input 事件的合写

```
1 <script setup>
2   import { ref } from 'vue'
3   const msg = ref('')
4 </script>
5 <template>
6   <p>{{ msg }}</p>
7   <input type="text" v-model="msg">
8   <input type="text" :value="msg" @input="msg = $event.target.value" >
9 </template>
```

1.3 注意

\$event 出现在模板中, 用来获取事件的形参

1.4 总结

1. v-model的原理是什么？

答：v-model只是个语法糖，作用在原生输入框上，本质是 `:value+@input` 的组合

```
1 <input v-model="msg"/>
2
3 <!-- 等同于 -->
4 <input type="text" :value="msg" @input="msg = $event.target.value" >
```

2. v-model用在组件上

2.1 分析

v-model除了用在表单元素上, 还可以用在组件上, 实现数据的双向绑定

2.2 需求

实现父组件和子组件数据的双向绑定

2.3 代码演示

BitSelect.vue

```
1 <script setup>
2   // 接收 modelValue
3   const props = defineProps({
4     modelValue: {
5       type: String,
6       default: ''
7     }
8   })
9
10  // 获取 emit
11  const emit = defineEmits()
12 </script>
13
14 <template>
15   <select
16     :value="props.modelValue"
17     @change="emit('update:modelValue', $event.target.value)">
18     <option value="111">北京</option>
19     <option value="222">上海</option>
20     <option value="333">深圳</option>
21     <option value="444">杭州</option>
22     <option value="555">苏州</option>
23   </select>
24 </template>
```

App.vue

```
1 <script setup>
2   import { ref } from 'vue'
3   import BitSelect from './components/BitSelect.vue'
```

```
4   const activeId = ref('222')
5 </script>
6
7 <template>
8   <BitSelect v-model="activeId"/>
9 </template>
```

2.4 简化代码

可以使用 `defineModel()` 进行简化上述代码

BitSelect.vue

```
1 <script setup>
2   const model = defineModel()
3 </script>
4
5 <template>
6   <select v-model="model">
7     <option value="111">北京</option>
8     <option value="222">上海</option>
9     <option value="333">深圳</option>
10    <option value="444">杭州</option>
11    <option value="555">苏州</option>
12  </select>
13 </template>
```

2.5 总结

1. 今后在组件上使用 v-model 的规则是？

答：1、父组件绑定 `v-model="数据"`

2、子组件 `const model = defineModel()` 接收数据，子组件可以直接操作model，即可实现数据双向绑定

二、ref

1. 作用

ref 可以用于 获取原生 DOM 元素 或 组件实例

2. 步骤

1. 声明并绑定 ref

2. 通过 ref.value 获取

3. 获取原生DOM

3.1 静态代码

components/MyChart.vue

```
1 <script setup>
2   // npm i echarts
3   import * as echarts from 'echarts'
4
5   const option = {
6     title: {
7       text: 'ECharts 入门案例',
8     },
9     xAxis: {
10      data: ['衬衫', '羊毛衫', '雪纺衫', '裤子', '高跟鞋', '袜子'],
11    },
12    yAxis: {},
13    series: [
14      {
15        name: '销量',
16        type: 'bar',
17        data: [5, 20, 36, 10, 10, 20],
18      }
19    ]
20  }
21 </script>
22
23 <template>
24   <div class="chart-box"></div>
25 </template>
26
27 <style scoped>
28   .chart-box {
29     width: 400px;
30     height: 300px;
31     margin: 100px auto;
32     border: 3px solid #000;
33     border-radius: 6px;
34   }
35 </style>
```

App.vue

```
1 <script setup>
```

```

2   import MyChart from './components/MyChart.vue'
3 </script>
4
5 <template>
6   <MyChart />
7 </template>
8
9 <style>
10 </style>

```

3.2 完整代码

MyChart.vue

```

1 <script setup>
2   import { ref } from 'vue'
3
4   import * as echarts from 'echarts'
5
6   const chartRef = ref(null)
7
8   const option = {
9     title: {
10       text: 'ECharts 入门案例',
11     },
12     xAxis: {
13       data: ['衬衫', '羊毛衫', '雪纺衫', '裤子', '高跟鞋', '袜子'],
14     },
15     yAxis: {},
16     series: [
17       {
18         name: '销量',
19         type: 'bar',
20         data: [5, 20, 36, 10, 10, 20],
21       }
22     ]
23   }
24
25   onMounted(() => {
26     const chart = echarts.init(chartRef.value)
27     chart.setOption(option)
28   })
29 </script>
30
31 <template>
32   <div class="chart-box" ref="chartRef"></div>

```

```
33 </template>
34
35 <style scoped>
36 .chart-box {
37   width: 400px;
38   height: 300px;
39   margin: 100px auto;
40   border: 3px solid #000;
41   border-radius: 6px;
42 }
43 </style>
```

4. 调用组件方法

4.1 静态代码

components/MyForm.vue

```
1 <script setup></script>
2 <template>
3   <div class="login-box">
4     账户: <input type="text" /><br /><br />
5     密码: <input type="password" /><br /><br />
6   </div>
7 </template>
8
```

App.vue

```
1 <script setup>
2   import MyForm from './components/MyForm.vue'
3 </script>
4
5 <template>
6   <MyForm />
7
8   <button>登 录</button>
9 </template>
10
11 <style>
12   #app {
13     width: 300px;
14     margin: 100px auto;
```

```
15   }  
16 </style>
```

4.2 完整代码

components/MyForm.vue

```
1 <script setup>  
2   // 表单校验  
3   const validate = () => {  
4     return Math.random() > 0.5 ? true : false  
5   }  
6  
7   // 暴露给组件，目的是父组件可以通过 ref 可以拿到子组件的方法  
8   defineExpose({  
9     validate  
10  })  
11 </script>  
12 <template>  
13   <div class="login-box">  
14     账户: <input type="text" /><br /><br />  
15     密码: <input type="password" /><br /><br />  
16   </div>  
17 </template>
```

App.vue

```
1 <script setup>  
2   import { ref } from 'vue'  
3   import MyForm from './components/MyForm.vue'  
4  
5   const formRef = ref(null)  
6  
7   // 登录  
8   const onLogin = () => {  
9     console.log(formRef.value)  
10  
11     // 进行校验  
12     if (formRef.value.validate()) {  
13       console.log('ok')  
14     } else {  
15       console.log('error')  
16     }  
17   }
```

```
18 </script>
19
20 <template>
21   <MyForm ref="formRef" />
22
23   <button @click="onLogin">登 录</button>
24 </template>
25
26 <style>
27   #app {
28     width: 300px;
29     margin: 100px auto;
30   }
31 </style>
```

三、nextTick

1. 需求

编辑标题, 编辑框自动聚焦

1. 点击编辑, 显示编辑框
2. 让编辑框, 立刻获取焦点



2. 代码实现

```
1 // 显示输入框
2 isShowEdit.value = true
3 // 获取焦点
4 inputRef.value.focus()
```

3. 问题

显示之后, 立刻获取焦点是不能成功的!

原因: `Vue` 是异步更新DOM (提升性能)

4. 解决方案

`nextTick`: 等DOM更新后, 才会触发执行此方法里的函数体

语法: `nextTick(函数体)`

```
1 nextTick(() => {  
2   inputRef.value.focus()  
3 })
```

比特就业课