第5章 深入了解 Vue 组件

5.1 什么是组件化开发

组件化开发是 Vue.js 框架的核心特性之一，也是目前前端技术框架中最常见的一种开发模式。在 Vue.js 中，组件就是一个可以复用的 Vue 实例，拥有独一无二的组件名称，可以扩展 HTML 元素，使用组件名称作为自定义的 HTML 标签。

在 Vue.js 项目中，每个组件都是一个 Vue 实例，所以组件内的属性选项都是相同的，例如 data、computed、watch、methods 及生命周期钩子等。仅有的例外是像 el 这样实例特有的选项。

在很多场景下，网页中的某些部分是可以复用的，例如头部导航、猜你喜欢、热点信息等。我们可以将网站中能够重复使用的部分设计成一个个组件，当需要的时候，直接引用这个组件即可。

Vue 组件化开发有别于前端传统的模块化开发。模块化是为了实现每个模块、方法的单一功能，一般通过代码逻辑进行划分，而组件化开发，更多的是实现前端 UI 的重复使用。

5.2 Vue 自定义组件

在使用 Vue CLI 工具创建的项目中，src 目录是用来存放项目源码的，在 src 目录下会自动创建两个子目录，一个是 src/views 目录，另一个是 src/components 目录。这两个子目录都是用来创建组件的，但是为了区分组件的功能，一般在 src/views 目录下创建的是视图组件，而在 src/components 目录下创建的是公共 UI 组件。

我们可以在 src/views 目录下创建一个 Home.vue 的视图组件，代码如下：

```
<template>
  <div>
    这是 Home 页面!
  </div>
</template>
```

然后将 Home.vue 组件引入项目的根组件 App.vue 中，代码如下：

```
<template>
  <div>
```

```
        <!-- 使用标签的方式使用自定义组件 -->
      <Home></Home>
    </div>
</template>

<script>
import Home from './views/Home.vue'
export default {
    components: {
            Home
    }
}
</script>
```

在 App.vue 根组件中使用 components 选项注册自定义组件，完成上面代码后启动项目，在浏览器中运行的效果如图 5.1 所示。

图 5.1　在浏览器中运行的效果

5.2.1　组件的封装

在项目的开发中，很多时候需要将某些 UI 部分封装成一个独立的组件，这部分 UI 可以是一个页面中的模块，例如商品列表，也可以是一个很小的部件，例如按钮。对于这种常用的 UI 元素可以创建一个组件并放到 components 目录下。

在 src/components 目录下新建一个 Button.vue 文件，代码如下：

```
<template>
    <button>按钮</button>
</template>
```

在上面的代码中，<template>标签内只能有一个子节点，如果在<button>标签的同级位置还有其他的标签元素，需要在<button>和同级标签的外部再添加一个父节点。

自定义 Button.vue 组件创建成功后，在 App.vue 根组件中引入，代码如下：

```
<template>
  <div>
      <!-- 用标签的方式使用自定义组件 -->
    <el-button />
  </div>
</template>
```

```
<script>
import Button from './components/Button.vue'
export default {
    components: {
            'el-button': Button
    }
}
</script>
```

在上面的代码中，components 选项内使用的 el-button 作为自定义组件的别名，在 <template>模板中用 <el-button> 作为 HTML 扩展标签。启动项目，在浏览器中运行的效果如图 5.2 所示。

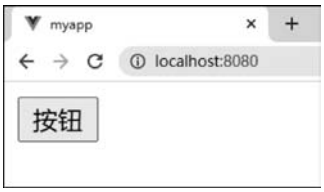

图 5.2　自定义按钮组件

5.2.2　自定义组件上的属性

在 5.2.1 节示例代码中，使用 <el-button> 自定义 UI 组件就可以在浏览器中渲染出按钮元素，但是在很多时候，自定义组件内部不能直接定义按钮的样式与属性，应该由组件的使用者对按钮组件进行创建并赋予其相关的属性值。我们可以在自定义组件上以使用标签属性的方式，为自定义组件传入参数，并在组件内部渲染。

例如，可以在创建 Button.vue 组件时，为其设置 props 选项属性，代码如下：

```
<template>
    <button>{{text}}</button>
</template>

<script>
export default {
    props: {
        text: String
    }
}
</script>
```

在上面的示例代码中，使用 props 选项属性为 Button.vue 自定义组件定义了属性 text，在其他组件内引入该自定义组件时，就可以使用该属性作为标签属性，并为自定义组件传入具体的内容。

在 App.vue 中引入 Button.vue 自定义组件，代码如下：

```
<template>
  <div>
      <!-- 用标签的方式使用自定义组件，并通过 text 属性设置按钮文本内容 -->
    <el-button text="提交" />
  </div>
</template>

<script>
import Button from './components/Button.vue'
export default {
    components: {
            'el-button': Button
    }
}
</script>
```

在上面的示例代码中，通过为 <el-button> 标签设置 text 属性的方式设置按钮显示的文本内容，在浏览器中运行的效果如图 5.3 所示。

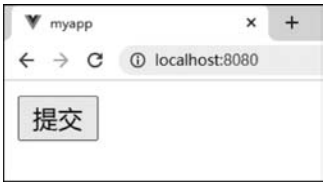

图 5.3　设置自定义按钮文本

5.2.3　自定义组件上的事件

自定义组件与预定义的 HTML 标签是有区别的，预定义的 HTML 标签能够被浏览器识别并渲染，所以会在浏览器中支持 HTML 标签上的原始属性和事件，但是自定义组件的标签在浏览器中渲染的是组件内部定义的 UI 样式，浏览器无法直接识别自定义组件标签上的属性和事件。

在使用自定义组件时，如果要为组件标签设置属性和事件，就要在自定义组件的内部提前声明。在组件上设置事件与设置属性还有所差别，设置属性只需要在组件内部使用 props 选项，而设置事件，需要考虑事件的执行时机，并在组件内部为原始的 HTML 标签定义该事件。

例如自定义按钮组件，并为组件设置单击事件，代码如下：

```
<template>
    <button @click="handleClick">{{text}}</button>
</template>
<script>
export default {
    props: {
        text: String
    },
    methods: {
        handleClick(){
```

```
            this.$emit('click')
        }
    }
}
</script>
```

上面的示例代码为原生的 <button> 标签添加了单击事件，并在单击事件的触发函数中调用了 $emit()方法，触发该自定义组件定义的名为 click 的事件。

在 App.vue 中引入 Button.vue 组件，代码如下：

```
<template>
  <div>
      <!-- 用标签的方式使用自定义组件 -->
    <el-button text="提交" @click="submit"/>
  </div>
</template>
<script>
import Button from './components/Button.vue'
export default {
    components: {
        'el-button': Button
    },
    methods: {
        submit(){
            console.log('提交按钮被单击了')
        }
    }
}
</script>
```

在上面的示例代码中，为<el-button>标签定义了@click 事件，在自定义组件标签中没有默认的单击事件，但是在创建 Button.vue 组件时，为原生<button>按钮元素添加了单击事件，并使用 $emit('click') 方法触发了 click 事件，所以在<el-button>标签上定义@click 事件是可以被触发的。

在浏览器中运行的效果如图 5.4 所示。

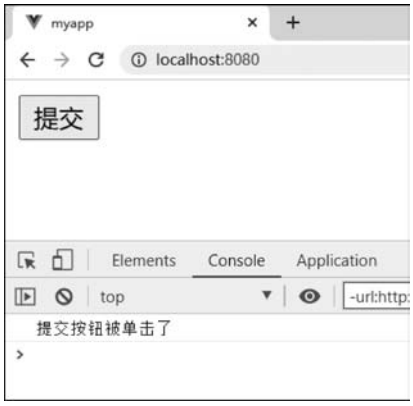

图 5.4　自定义单击事件触发效果

5.3 组件属性校验

在自定义组件时，可以在自定义组件文件的内部使用 props 选项来为组件自定义一些功能。在使用自定义组件标签时，这些预先定义的 props 属性可以使用标签属性的书写方式在组件标签上声明，此时，标签上的属性值被传入自定义组件内的 props 选项中，在组件内就可以获取这个值了。

HTML 标签中的属性名对大小写不敏感，所以浏览器会把所有大写字符解释为小写字符。这意味着当使用 DOM 中的模板时，camelCase（驼峰命名法）的 prop 名需要使用其等价的 kebab-case（短横线分隔命名）命名，代码如下：

```
Vue.component('blog-post', {
  //在 JavaScript 中为 camelCase
  props: ['postTitle'],
  template: '<h3>{{ postTitle }}</h3>'
})
```

在 HTML 中属性名使用 kebab-case 格式命名，代码如下：

```
<blog-post post-title="hello!"></blog-post>
```

我们可以为组件的 prop 指定验证要求，如果有一个需求没有被满足，则 Vue 会在浏览器控制台中报出警告。在使用 prop 验证属性的类型时，值的类型应使用对象，而不能使用字符串，代码如下：

```
Vue.component('my-component', {
  props: {
    //基础的类型检查 ('null'和'undefined'会通过任何类型验证)
    propA: Number,
    //多个可能的类型
    propB: [String, Number],
    //必填的字符串
    propC: {
      type: String,
      required: true
    },
    //带有默认值的数字
    propD: {
      type: Number,
      default: 100
    },
    //带有默认值的对象
    propE: {
      type: Object,
      //对象或数组默认值必须从一个工厂函数获取
      default: function () {
```

```
        return { message: 'hello' }
      }
    },
    //自定义验证函数
    propF: {
      validator: function (value) {
        //这个值必须匹配下列字符串中的一个
        return ['success', 'warning', 'danger'].indexOf(value) !== -1
      }
    }
  }
})
```

33min

5.4 组件通信

组件化开发是 Vue 中的核心概念之一，通过设计具有各自状态的 UI 组件，然后由这些组件拼成更加复杂的 UI 页面，使代码更加简洁、容易维护。创建自定义组件在 Vue 开发中是非常常见的，在这种开发场景下必定会涉及组件之间的通信。在本节中将要学习的是如何实现组件之间的数据交互。

5.4.1 父组件向子组件通信

因为 Vue 项目采用单向数据流，所以只能从父组件将数据传递给子组件。具体传递的方式，可以使用 5.2.2 节讲到的 props 选项。在子组件的 props 选项中定义属性，在父组件中就可以向子组件传递值了。

子组件 Son.vue 文件代码如下：

```
<template>
    <div>
        子组件接收父组件传值：{{text}}
    </div>
</template>
<script>
export default {
    props: {
        text: String
    }
}
</script>
```

父组件 Father.vue 文件代码如下：

```
<template>
    <div>
        <h3>父组件向子组件传值</h3>
```

```
        <son v-bind:text="msg"></son>
    </div>
</template>
<script>
import Son from './Son.vue'
export default {
    components: {
        'son': Son
    },
    data(){
        return {
            msg: 'hello world!'
        }
    }
}
</script>
```

在浏览器中运行的效果如图 5.5 所示。

图 5.5　父组件向子组件传值

5.4.2　子组件向父组件通信

单向数据流决定了父组件可以影响子组件的数据，但是反之不行。子组件内数据发生更新后，在父组件中无法直接获取更新后的数据。要想实现子组件向父组件传递数据，可以在子组件数据发生变化后，触发一个事件方法，然后由这个事件方法告诉父组件数据更新了。在父组件中只需对这个事件进行监听，当捕获到这个事件运行后，再对父组件的数据进行同步更新。

子组件 Son.vue 文件代码如下：

```
<template>
    <div>
        子组件输入新值:
        <input type="text" v-model="value">
        <button @click="submit">提交</button>
    </div>
</template>
<script>
export default {
```

```
    data(){
        return {
            value: ''
        }
    },
    methods: {
        submit(){
            this.$emit('show',this.value);
        }
    }
}
</script>
```

父组件 Father.vue 文件代码如下：

```
<template>
    <div>
        <h3>父组件监听子组件数据更新：{{msg}}</h3>
        <son v-on:show="showMsg"></son>
    </div>
</template>
<script>
import Son from './Son.vue'
export default {
    components: {
        'son': Son
    },
    data(){
        return {
            msg: ''
        }
    },
    methods: {
        showMsg(msg){
            this.msg = msg
        }
    }
}
</script>
```

在上面的示例代码中，父组件中使用 v-on 事件监听器来监听子组件的事件，在子组件中使用 $emit() 触发当前实例上的事件。在浏览器中运行的效果如图 5.6 所示。

图 5.6　子组件向父组件传递数据

5.5 插　　槽

在构建页面时，我们常常会把具有公共特性的部分抽取出来，封装成一个独立的组件，但是在实际使用过程中又会产生一些其他问题，不能完全满足开发的需求。例如，当我们需要在自定义组件内添加一些新的元素时，原来组件的封装方式不能实现。这时，我们就需要使用插槽来分发内容。

2min

5.5.1 什么是插槽

Vue 为了实现组件的内容分发，在组件的相关内容中提供了一套用于组件内容分发的 API，也就是插槽。这套 API 使用 < slot > 内置组件作为承载分发内容的出口，代码如下：

创建父组件 Demo. vue，代码如下：

```
<template>
    <div>
        <h3>在父组件中使用插槽</h3>
        <my-slot>
            <p>这是父组件中添加的元素</p>
        </my-slot>
    </div>
</template>
<script>
import Myslot from './Myslot.vue'
export default {
    components: {
        'my-slot': Myslot
    }
}
</script>
```

创建子组件 Myslot. vue，代码如下：

```
<template>
    <div>
        <p>这是子组件内容</p>
        <slot></slot>
    </div>
</template>
```

在浏览器中运行的效果如图 5.7 所示。

5.5.2 具名插槽

在 Vue 2.6 中，具名插槽和作用域插槽引入了一个新的统一语法的 v-slot 指令。它取代了 slot 和 slot-scope 特性。

图 5.7　插槽效果

在实际的开发过程中，组件中的插槽不止一个，有时需要多个插槽，代码如下：

```
<div>
    <div class="header">
        这是页面头部
    </div>
    <div>
        这是页面的主体内容
    </div>
    <div>
        这是页面的底部
    </div>
</div>
```

针对上面的示例，<slot>元素有一个 name 属性，这个属性可以用来定义额外的插槽，代码如下：

```
<div>
    <div class="header">
        <slot name="header"></slot>
    </div>
    <div>
        <slot></slot>
    </div>
    <div>
        <slot name="footer"></slot>
    </div>
</div>
```

当 <slot> 元素上没有定义 name 属性时，<slot> 出口会带有隐含的 default。在向具名插槽提供内容时，可以在一个 <template> 元素上使用 v-slot 指令，并以 v-slot 的参数形式提供其名称，代码如下：

```
<base-layout>
    <template v-slot:header>
        这是页面头部
    </template>
```

```
    <p>这是页面主体部分</p>
    <template v-slot:footer>
        这是页面底部
    </template>
</base-layout>
```

在上面的代码中，所有的内容都被传入对应的插槽，没有使用带有 v-slot 的< template >中的内容会被视为默认插槽的内容。

v-slot 指令与 v-on、v-bind 类似，也有自己的缩写形式，把 v-slot 替换为字符 # 即可，代码如下：

```
<base-layout>
    <template #header>
        这是页面头部
    </template>
    <p>这是页面主体部分</p>
    <template #footer>
        这是页面底部
    </template>
</base-layout>
```

5.5.3 作用域插槽

在使用插槽时，经常会有这样的应用场景：在父组件中定义的数据需要在子组件中也能访问。例如，在父组件中获取的商品列表数据，当父组件中引入商品卡片的组件时，需要在商品卡片组件内部使用插槽，并把商品数据传给子组件进行渲染。

例如，创建商品卡片组件 card.vue 文件，代码如下：

```
<template>
    <div>
        <slot>{{ goods.title }}</slot>
    </div>
</template>
```

在商品卡片组件中想要显示商品的标题，但是商品数据是在父组件中获取的，所以只有在父组件中使用 <card> 组件标签才可以访问 goods 对象。为了让 goods 在父级的插槽内容可用，可以将 goods 作为 <slot> 元素的一个属性绑定上去。

card.vue 文件代码如下：

```
<template>
    <div>
        <slot v-bind:goods="goods">
            {{ goods.title }}
        </slot>
    </div>
</template>
```

绑定在 < slot > 元素上的属性被称为插槽 prop。现在，在父级作用域中，可以给 v-slot 赋一个值，来定义提供的插槽 prop 的名字，代码如下：

```
<div>
    <card>
      <template v-slot:default="slotProps">
            {{ slotProps.goods.title }}
      </template>
    </card>
</div>
```