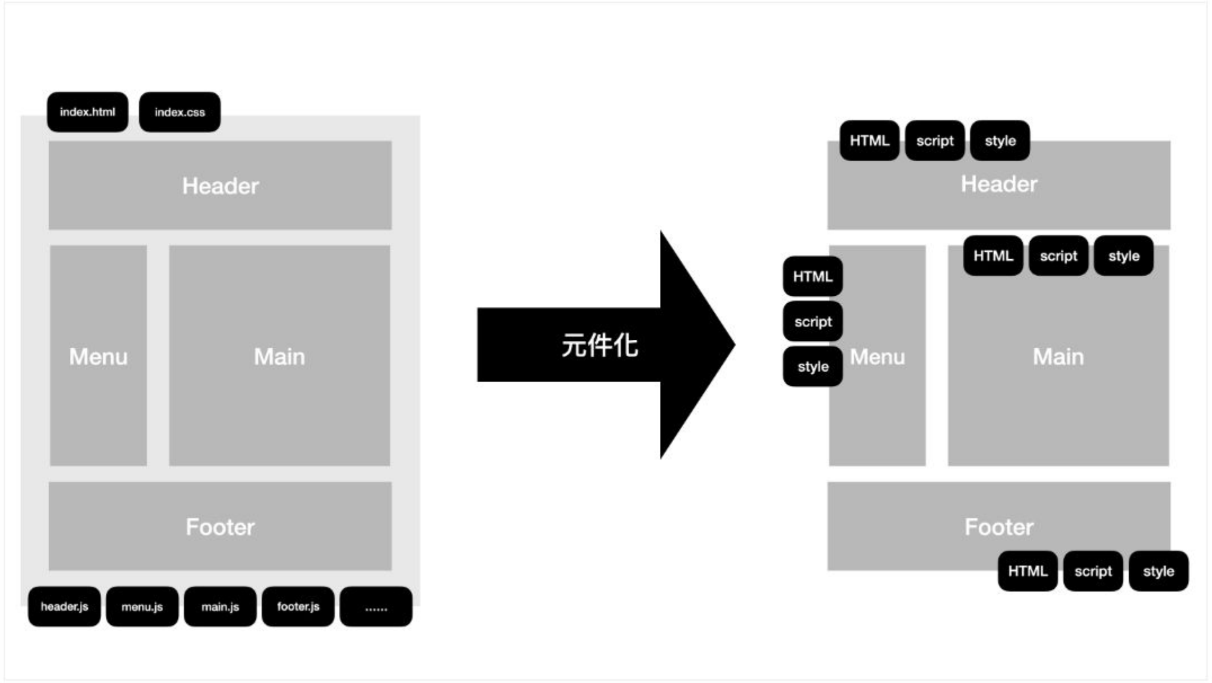
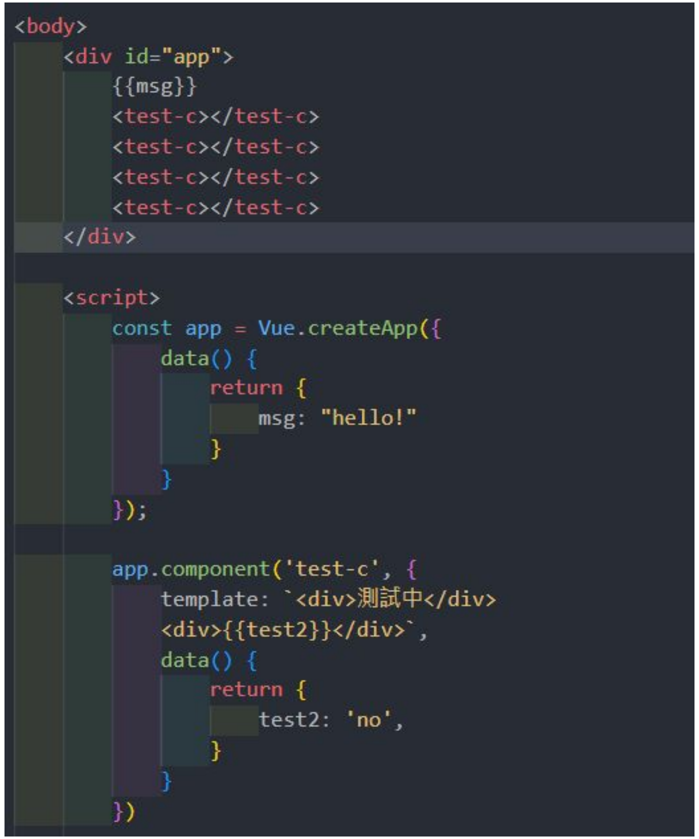
2.1 元件系統的特性

概念上與 VS 的 partialView 很像

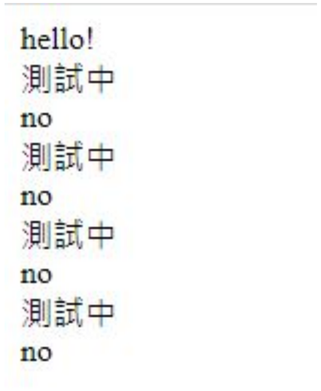


1. 封裝好的 UI
2. 功能重複使用
3. 元件中可含有元件

命名建議採用連字號



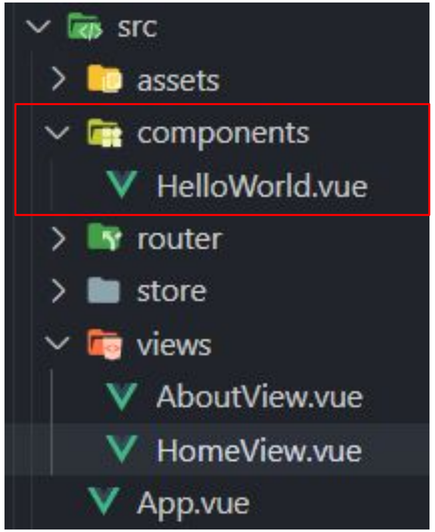
```
<body>
    <div id="app">
        {{msg}}
        <test-c></test-c>
        <test-c></test-c>
        <test-c></test-c>
        <test-c></test-c>
    </div>

    <script>
        const app = Vue.createApp({
            data() {
                return {
                    msg: "hello!"
                }
            }
        });

        app.component('test-c', {
            template: `<div>測試中</div>
            <div>{{test2}}</div>`,
            data() {
                return {
                    test2: 'no',
                }
            }
        })
```

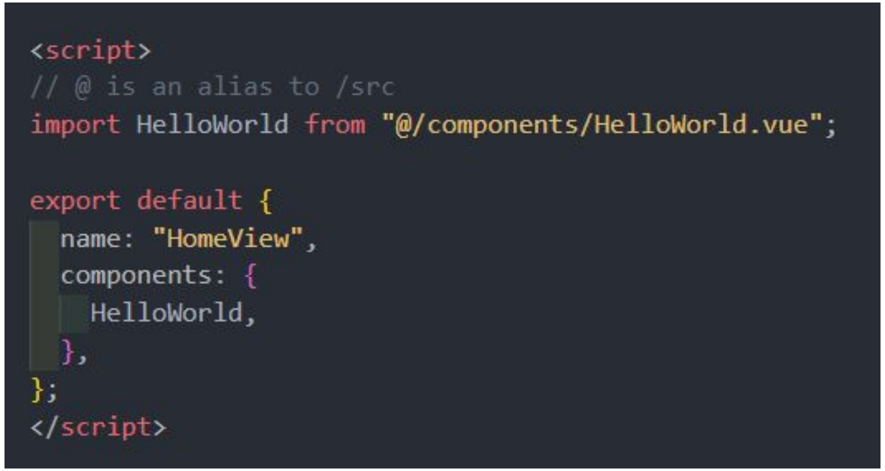


```
hello!
測試中
no
測試中
no
測試中
no
測試中
no
```

單一元件檔(Single Ffile Components)

將元件以 .vue 檔案包起來再透過 import 方式引入作為子元件

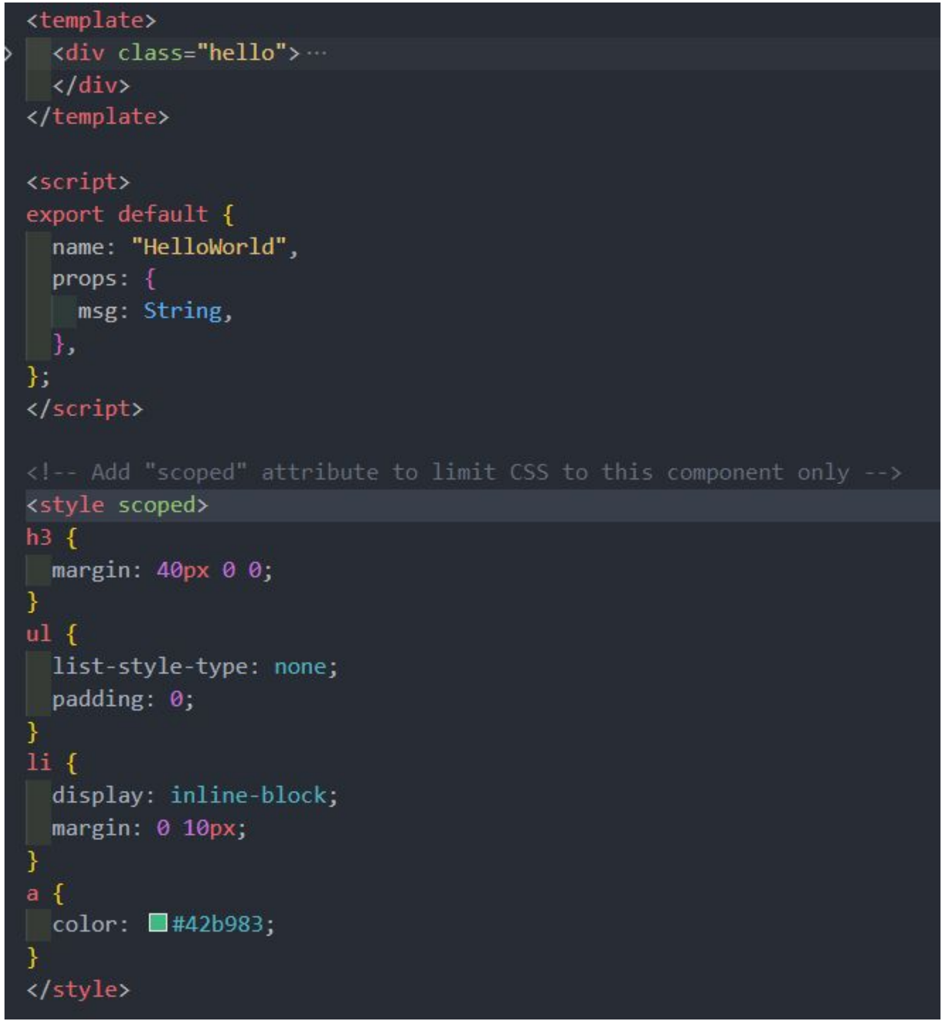




```
<script>
// @ is an alias to /src
import HelloWorld from "@/components/HelloWorld.vue";

export default {
  name: "HomeView",
  components: {
    HelloWorld,
  },
};
</script>
```

SFC 包含三部分:

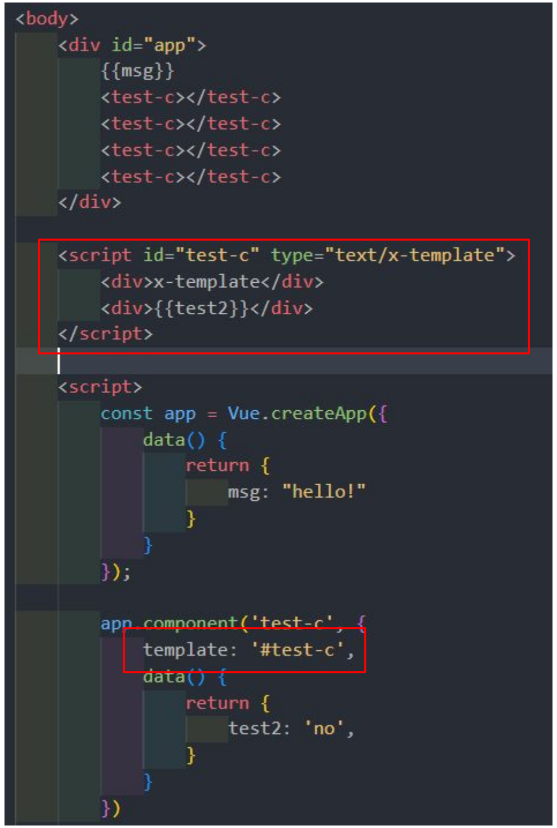
1. HTML 模板 <template>
2. 定義元件結構與邏輯的<script>
3. CSS 樣式的 <style>



```
<template>
  <div class="hello">⋯
  </div>
</template>

<script>
export default {
  name: "HelloWorld",
  props: {
    msg: String,
  },
};
</script>

<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>
h3 {
  margin: 40px 0 0;
}
ul {
  list-style-type: none;
  padding: 0;
}
li {
  display: inline-block;
  margin: 0 10px;
}
a {
  color: 🟩#42b983;
}
</style>
```

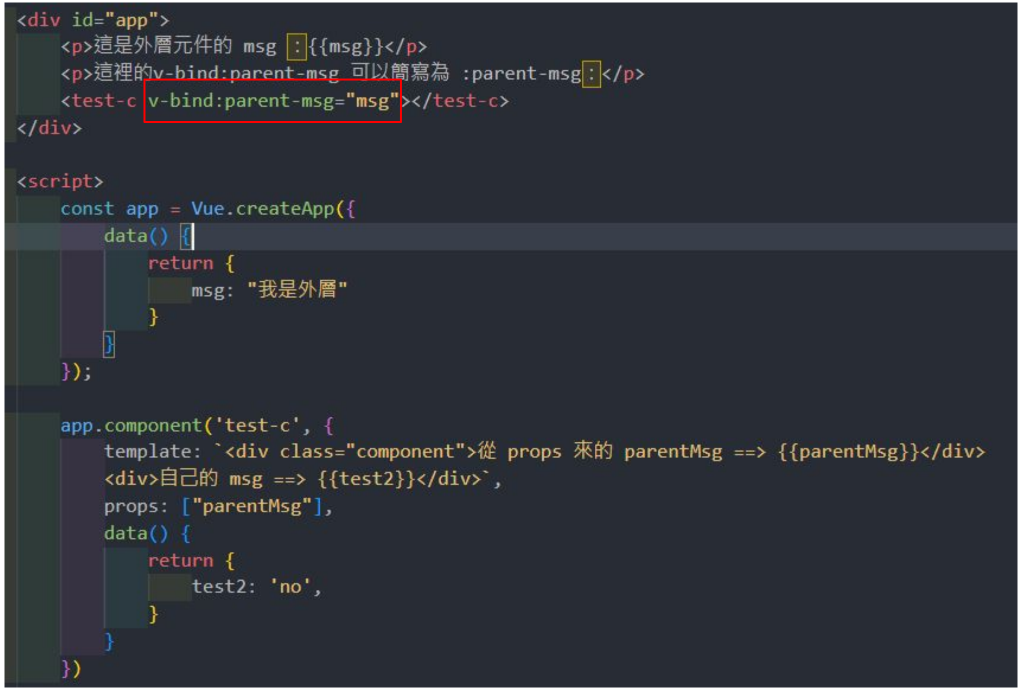
x-template 封裝模板

template 又臭又長寫在` ` 裡面真的有夠難看懂！

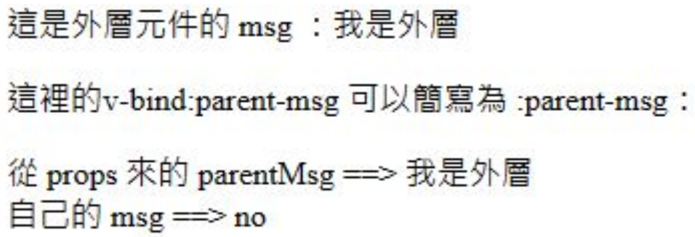


```
<body>
    <div id="app">
        {{msg}}
        <test-c></test-c>
        <test-c></test-c>
        <test-c></test-c>
        <test-c></test-c>
    </div>

    <script id="test-c" type="text/x-template">
        <div>x-template</div>
        <div>{{test2}}</div>
    </script>

    <script>
        const app = Vue.createApp({
            data() {
                return {
                    msg: "hello!"
                }
            }
        });

        app.component('test-c', {
            template: '#test-c',
            data() {
                return {
                    test2: 'no',
                }
            }
        })
```

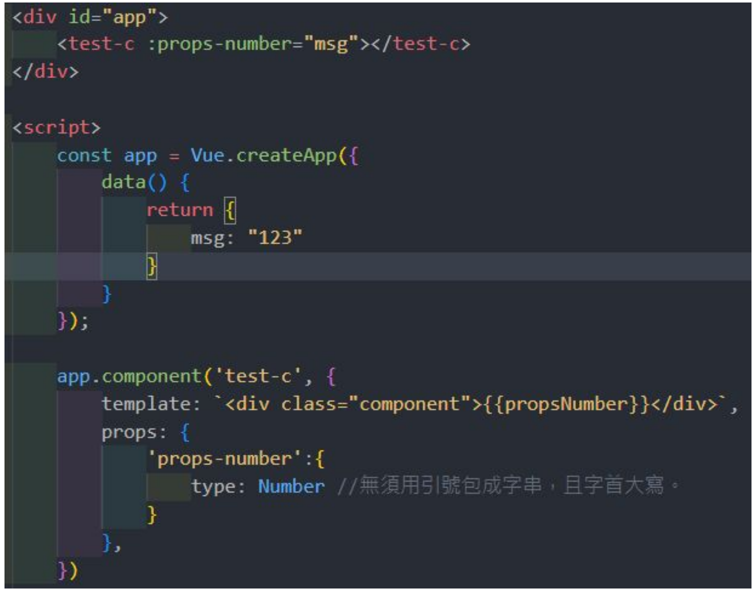
2.2 元件之間的溝通傳遞



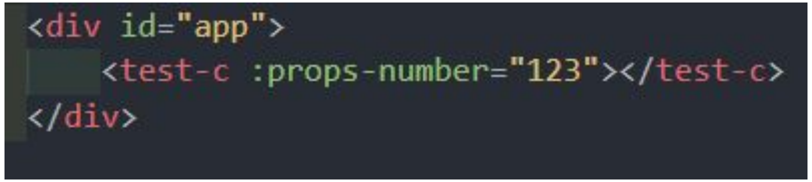
```
<div id="app">
    <p>這是外層元件的 msg :{{msg}}</p>
    <p>這裡的v-bind:parent-msg 可以簡寫為 :parent-msg:</p>
    <test-c v-bind:parent-msg="msg"></test-c>
</div>

<script>
    const app = Vue.createApp({
        data() {
            return {
                msg: "我是外層"
            }
        }
    });

    app.component('test-c', {
        template: `<div class="component">從 props 來的 parentMsg ==> {{parentMsg}}</div>
        <div>自己的 msg ==> {{test2}}</div>`,
        props: ["parentMsg"],
        data() {
            return {
                test2: 'no',
            }
        }
    })
```



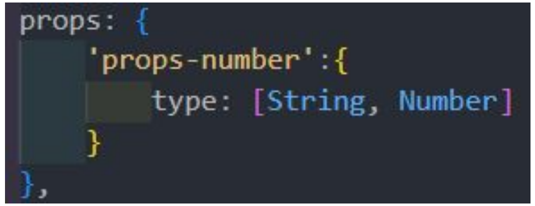
```
這是外層元件的 msg ：我是外層

這裡的v-bind:parent-msg 可以簡寫為 :parent-msg：

從 props 來的 parentMsg ==> 我是外層
自己的 msg ==> no
```



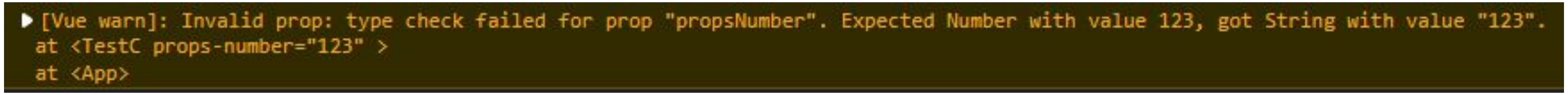
```
<div id="app">
    <test-c :props-number="msg"></test-c>
</div>

<script>
    const app = Vue.createApp({
        data() {
            return {
                msg: "123"
            }
        }
    });

    app.component('test-c', {
        template: `<div class="component">{{propsNumber}}</div>`,
        props: {
            'props-number':{
                type: Number //無須用引號包成字串，且字首大寫。
            }
        },
    })
```



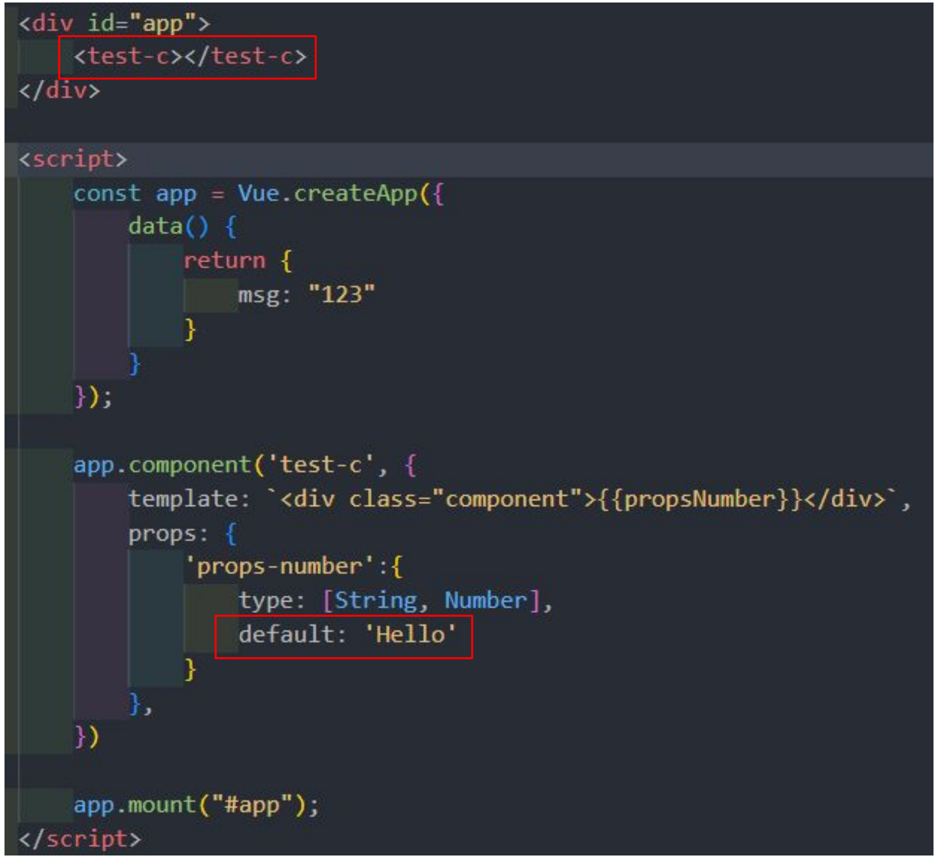
```
<div id="app">
    <test-c :props-number="123"></test-c>
</div>
```



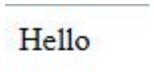
```
props: {
    'props-number':{
        type: [String, Number]
    }
},
```



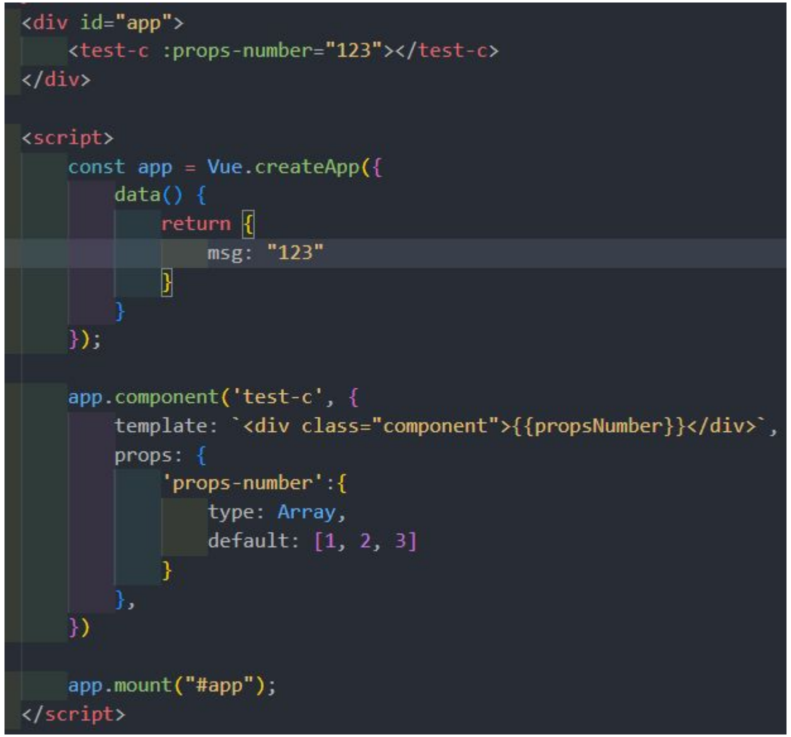
```
▶ [Vue warn]: Invalid prop: type check failed for prop "propsNumber". Expected Number with value 123, got String with value "123".
  at <TestC props-number="123" >
  at <App>
```



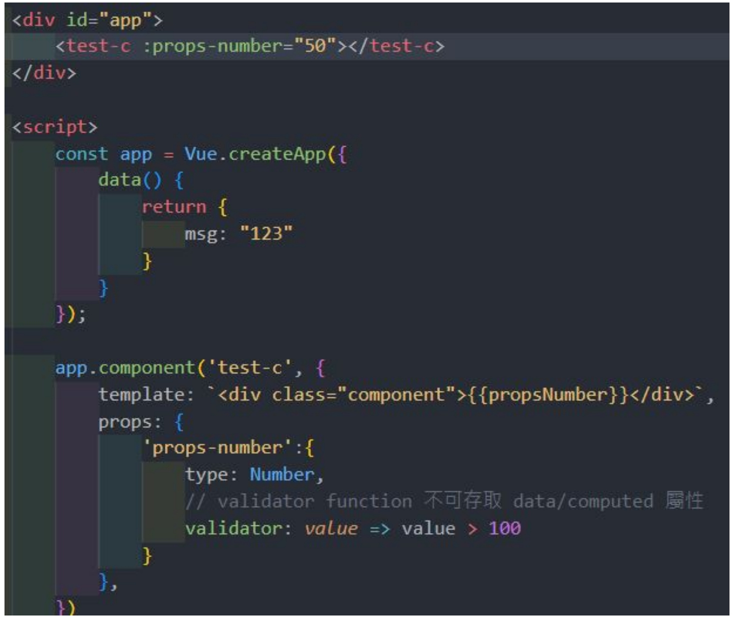
```
<div id="app">
    <test-c></test-c>
</div>

<script>
    const app = Vue.createApp({
        data() {
            return {
                msg: "123"
            }
        }
    });

    app.component('test-c', {
        template: `<div class="component">{{propsNumber}}</div>`,
        props: {
            'props-number':{
                type: [String, Number],
                default: 'Hello'
            }
        },
    })

    app.mount("#app");
</script>
```



```
Hello
```

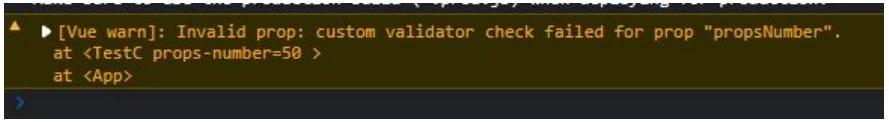


```
<div id="app">
    <test-c :props-number="123"></test-c>
</div>

<script>
    const app = Vue.createApp({
        data() {
            return {
                msg: "123"
            }
        }
    });

    app.component('test-c', {
        template: `<div class="component">{{propsNumber}}</div>`,
        props: {
            'props-number':{
                type: Array,
                default: [1, 2, 3]
            }
        },
    })

    app.mount("#app");
</script>
```

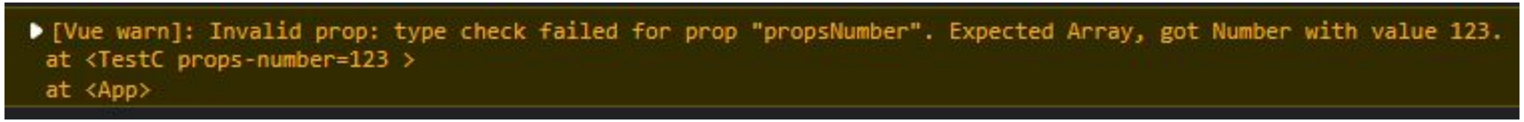


```
<div id="app">
    <test-c :props-number="50"></test-c>
</div>

<script>
    const app = Vue.createApp({
        data() {
            return {
                msg: "123"
            }
        }
    });

    app.component('test-c', {
        template: `<div class="component">{{propsNumber}}</div>`,
        props: {
            'props-number':{
                type: Number,
                // validator function 不可存取 data/computed 屬性
                validator: value => value > 100
            }
        },
    })
```

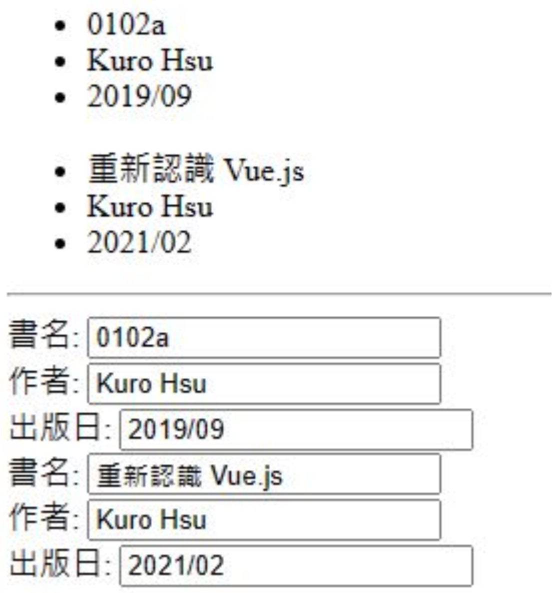
50



```
▶ [Vue warn]: Invalid prop: custom validator check failed for prop "propsNumber".
  at <TestC props-number=50 >
  at <App>
```



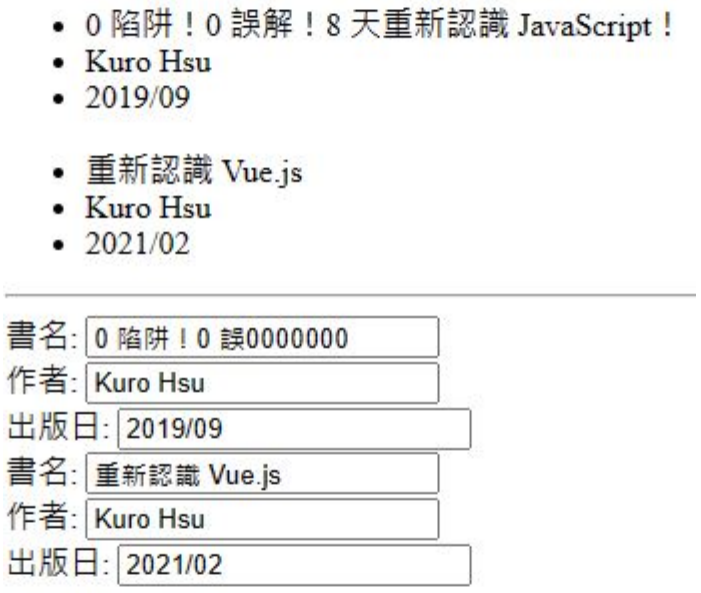
```
▶ [Vue warn]: Invalid prop: type check failed for prop "propsNumber". Expected Array, got Number with value 123.
  at <TestC props-number=123 >
  at <App>
```



- 0102a
- Kuro Hsu
- 2019/09

- 重新認識 Vue.js
- Kuro Hsu
- 2021/02

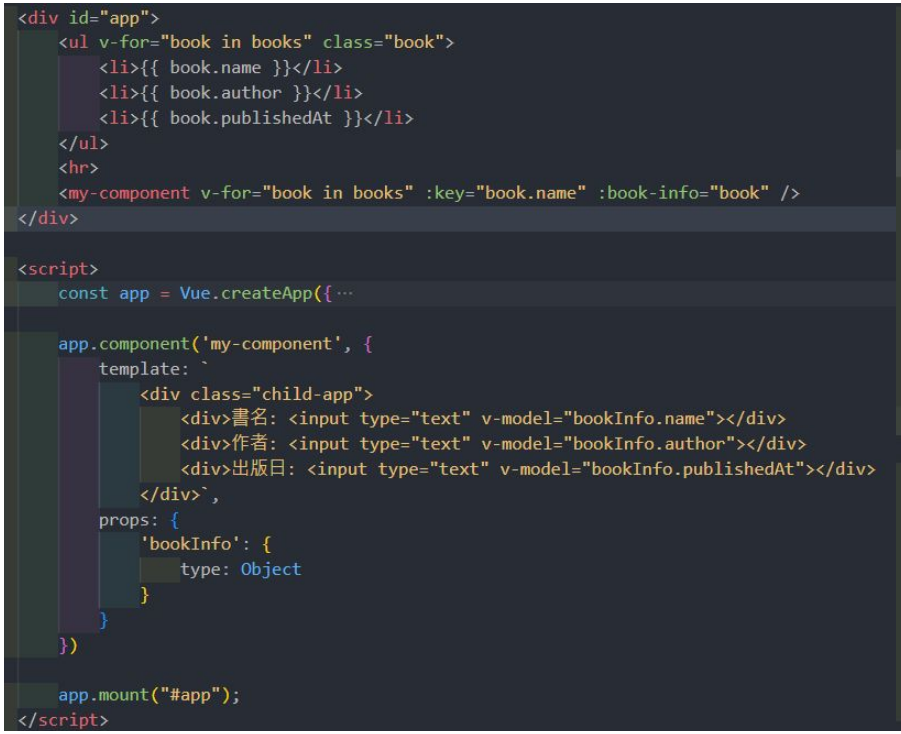
書名: 0102a
作者: Kuro Hsu
出版日: 2019/09
書名: 重新認識 Vue.js
作者: Kuro Hsu
出版日: 2021/02



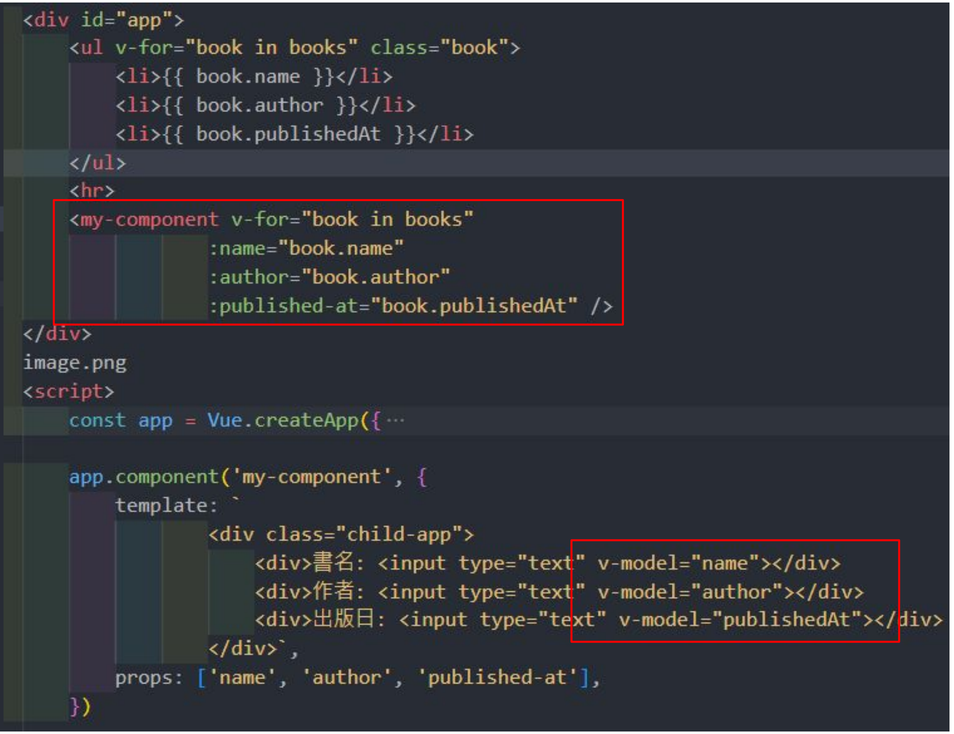
- 0 陷阱！0 誤解！8 天重新認識 JavaScript！
- Kuro Hsu
- 2019/09

- 重新認識 Vue.js
- Kuro Hsu
- 2021/02

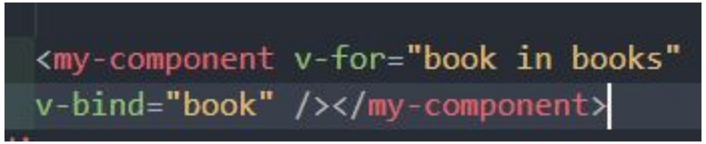
書名: 0 陷阱！0 誤0000000
作者: Kuro Hsu
出版日: 2019/09
書名: 重新認識 Vue.js
作者: Kuro Hsu
出版日: 2021/02



```
<div id="app">
    <ul v-for="book in books" class="book">
        <li>{{ book.name }}</li>
        <li>{{ book.author }}</li>
        <li>{{ book.publishedAt }}</li>
    </ul>
    <hr>
    <my-component v-for="book in books" :key="book.name" :book-info="book" />
</div>

<script>
    const app = Vue.createApp({…

    app.component('my-component', {
        template: `
            <div class="child-app">
                <div>書名: <input type="text" v-model="bookInfo.name"></div>
                <div>作者: <input type="text" v-model="bookInfo.author"></div>
                <div>出版日: <input type="text" v-model="bookInfo.publishedAt"></div>
            </div>`,
        props: {
            'bookInfo': {
                type: Object
            }
        }
    })

    app.mount("#app");
</script>
```

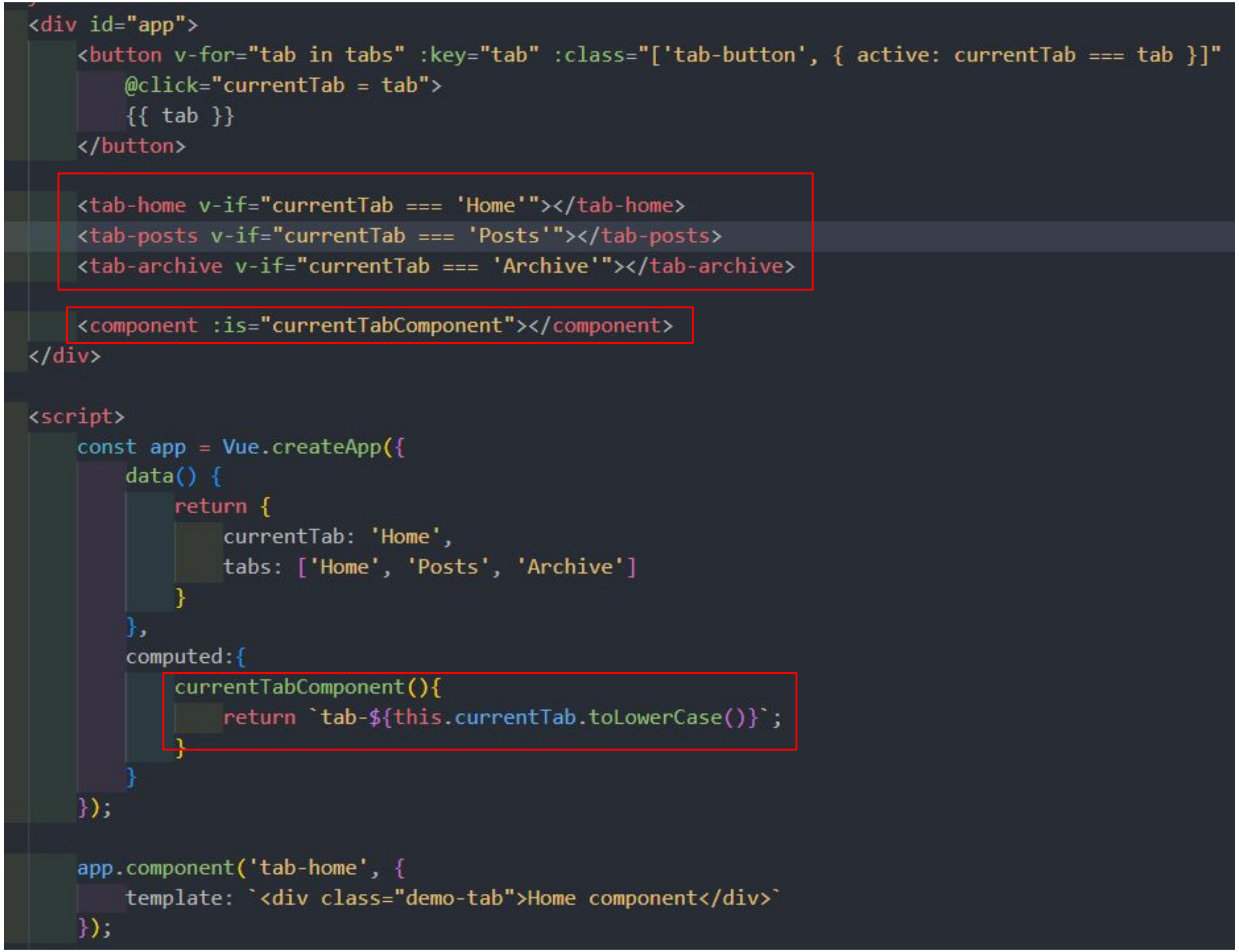


```
<div id="app">
    <ul v-for="book in books" class="book">
        <li>{{ book.name }}</li>
        <li>{{ book.author }}</li>
        <li>{{ book.publishedAt }}</li>
    </ul>
    <hr>
    <my-component v-for="book in books"
                  :name="book.name"
                  :author="book.author"
                  :published-at="book.publishedAt" />
</div>
image.png
<script>
    const app = Vue.createApp({…

    app.component('my-component', {
        template: `
            <div class="child-app">
                <div>書名: <input type="text" v-model="name"></div>
                <div>作者: <input type="text" v-model="author"></div>
                <div>出版日: <input type="text" v-model="publishedAt"></div>
            </div>`,
        props: ['name', 'author', 'published-at'],
    })
```



```
<my-component v-for="book in books"
v-bind="book" /></my-component>
```

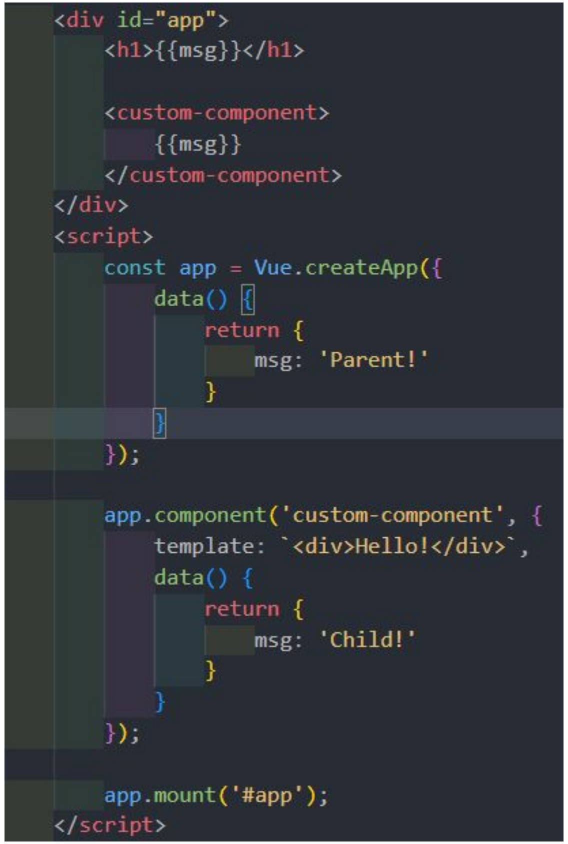
2.3 動態元件管理



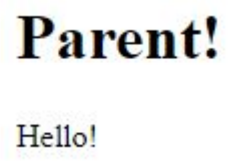
```
<div id="app">
    <button v-for="tab in tabs" :key="tab" :class="['tab-button', { active: currentTab === tab }]"
        @click="currentTab = tab">
        {{ tab }}
    </button>

    <tab-home v-if="currentTab === 'Home'"></tab-home>
    <tab-posts v-if="currentTab === 'Posts'"></tab-posts>
    <tab-archive v-if="currentTab === 'Archive'"></tab-archive>

    <component :is="currentTabComponent"></component>
</div>

<script>
    const app = Vue.createApp({
        data() {
            return {
                currentTab: 'Home',
                tabs: ['Home', 'Posts', 'Archive']
            }
        },
        computed:{
            currentTabComponent(){
                return `tab-${this.currentTab.toLowerCase()}`;
            }
        }
    });

    app.component('tab-home', {
        template: `<div class="demo-tab">Home component</div>`
    });
```

2.4 編譯作用與插槽

2.4.1 元件的編譯作用域



```
<div id="app">
    <h1>{{msg}}</h1>

    <custom-component>
        {{msg}}
    </custom-component>
</div>
<script>
    const app = Vue.createApp({
        data() {
            return {
                msg: 'Parent!'
            }
        }
    });

    app.component('custom-component', {
        template: `<div>Hello!</div>`,
        data() {
            return {
                msg: 'Child!'
            }
        }
    });

    app.mount('#app');
</script>
```



Parent!

Hello!

custom-component 裡的 {{ msg }} 自動被 template 裡的內容取代

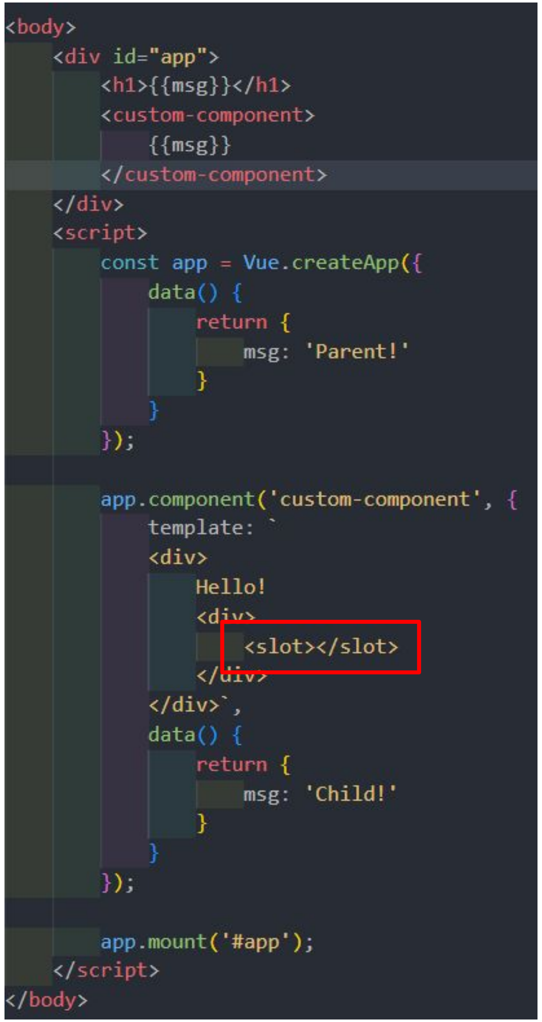
編譯元件模板時:

元件模板所定義 內容為主

編譯網頁模板時:

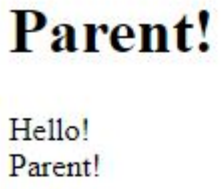
無視 custom-component 裡任何內容

以子元件模板取代



```
<body>
    <div id="app">
        <h1>{{msg}}</h1>
        <custom-component>
            {{msg}}
        </custom-component>
    </div>
    <script>
        const app = Vue.createApp({
            data() {
                return {
                    msg: 'Parent!'
                }
            }
        });

        app.component('custom-component', {
            template: `
            <div>
                Hello!
                <div>
                    <slot></slot>
                </div>
            </div>`,
            data() {
                return {
                    msg: 'Child!'
                }
            }
        });

        app.mount('#app');
    </script>
</body>
```

2.4.2 插槽(Slots)



```
Parent!

Hello!
Parent!
```

custom-component 裡的 {{ msg }} 取代為父層的 msg

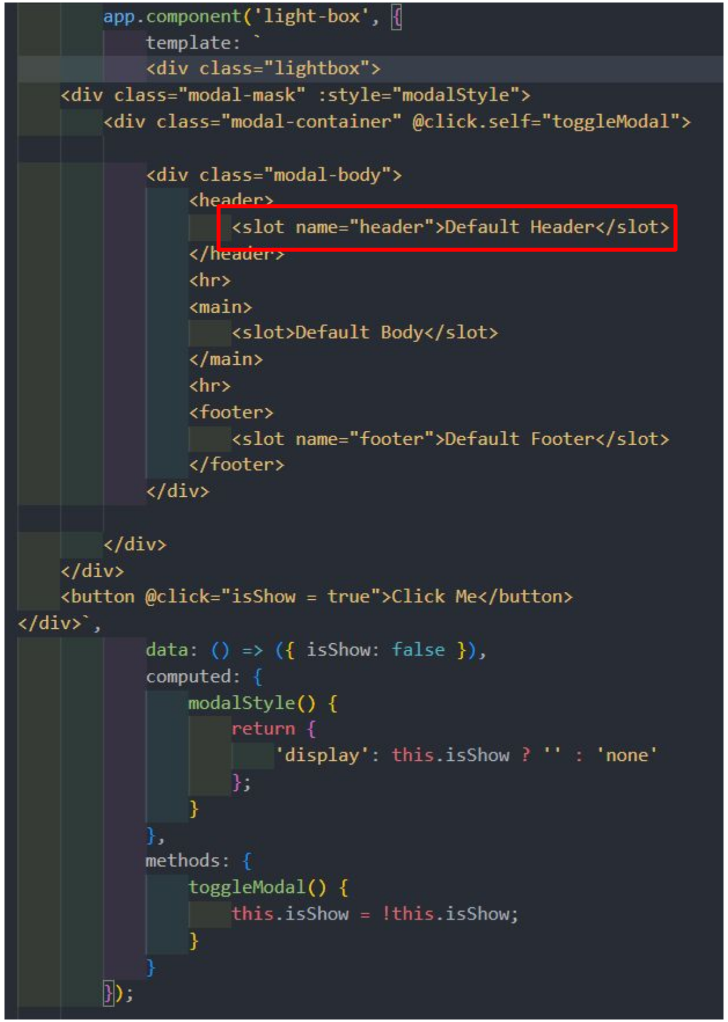
slot 特性:

保留空間傳入外部 內容

子元件對其無控制權

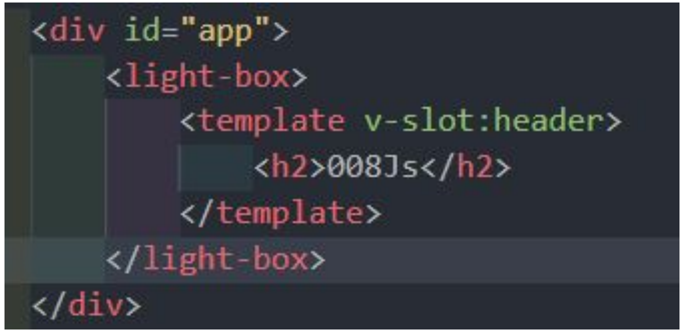
Q:於 slot 區域出現預設 內容?

A:Slot 預設內容

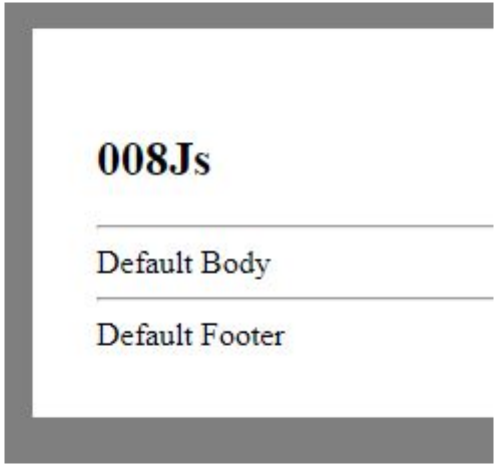


```
app.component('light-box', {
    template: `
    <div class="lightbox">
  <div class="modal-mask" :style="modalStyle">
    <div class="modal-container" @click.self="toggleModal">

      <div class="modal-body">
        <header>
          <slot name="header">Default Header</slot>
        </header>
        <hr>
        <main>
          <slot>Default Body</slot>
        </main>
        <hr>
        <footer>
          <slot name="footer">Default Footer</slot>
        </footer>
      </div>

    </div>
  </div>
  <button @click="isShow = true">Click Me</button>
</div>`,
      data: () => ({ isShow: false }),
      computed: {
        modalStyle() {
          return {
            'display': this.isShow ? '' : 'none'
          };
        }
      },
      methods: {
        toggleModal() {
          this.isShow = !this.isShow;
        }
      }
    });
```

2.4.3 具名插槽(Named Slots)



```
<div id="app">
  <light-box>
    <template v-slot:header>
      <h2>008Js</h2>
    </template>
  </light-box>
</div>
```

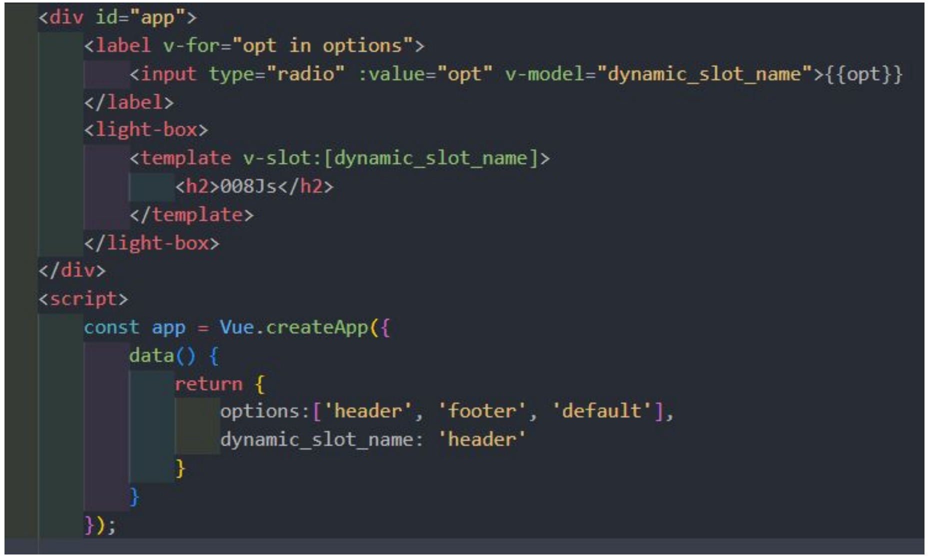


008Js

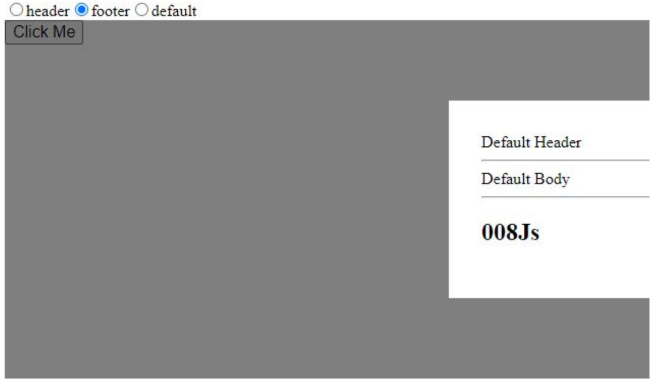
Default Body

Default Footer

2.4.3 動態切換具名插槽



```
<div id="app">
    <label v-for="opt in options">
        <input type="radio" :value="opt" v-model="dynamic_slot_name">{{opt}}
    </label>
    <light-box>
        <template v-slot:[dynamic_slot_name]>
            <h2>008Js</h2>
        </template>
    </light-box>
</div>
<script>
    const app = Vue.createApp({
        data() {
            return {
                options:['header', 'footer', 'default'],
                dynamic_slot_name: 'header'
            }
        }
    });
```



○ header ● footer ○ default

Click Me

Default Header

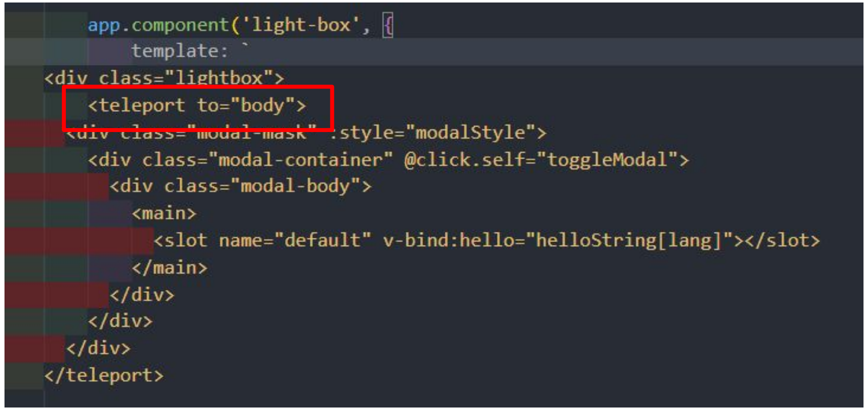
Default Body

008Js

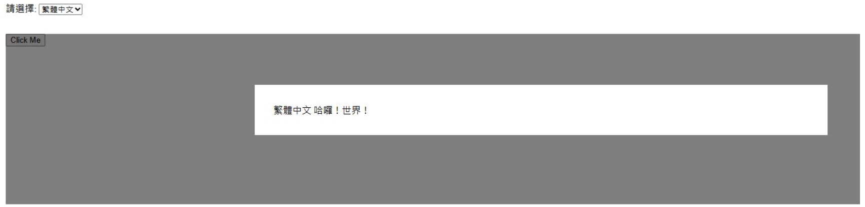
2.4.3 作用域插槽

2.4.4 teleport

將模板中特定的 DOM 移動至指定的位置渲染

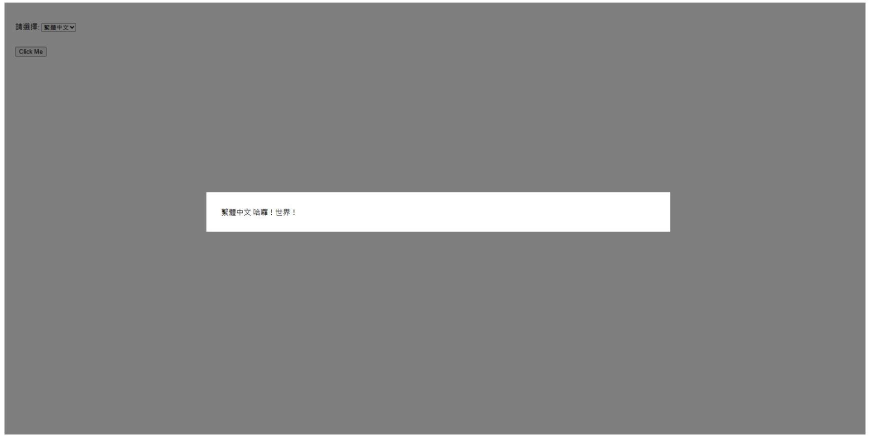


```
    app.component('light-box', {
        template: `
<div class="lightbox">
    <teleport to="body">
  <div class="modal-mask" :style="modalStyle">
    <div class="modal-container" @click.self="toggleModal">
      <div class="modal-body">
        <main>
          <slot name="default" v-bind:hello="helloString[lang]"></slot>
        </main>
      </div>
    </div>
  </div>
</teleport>
```



請選擇: 繁體中文

繁體中文 哈囉！世界！

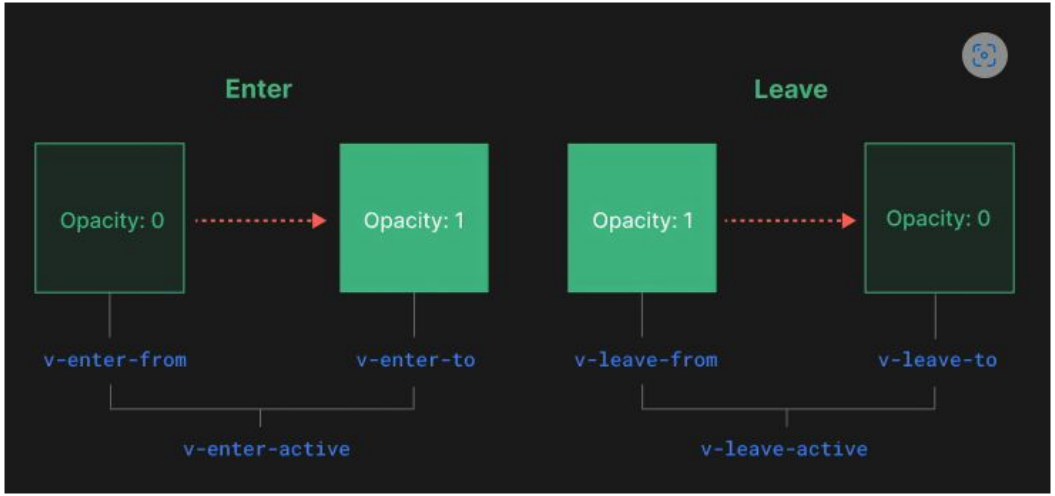


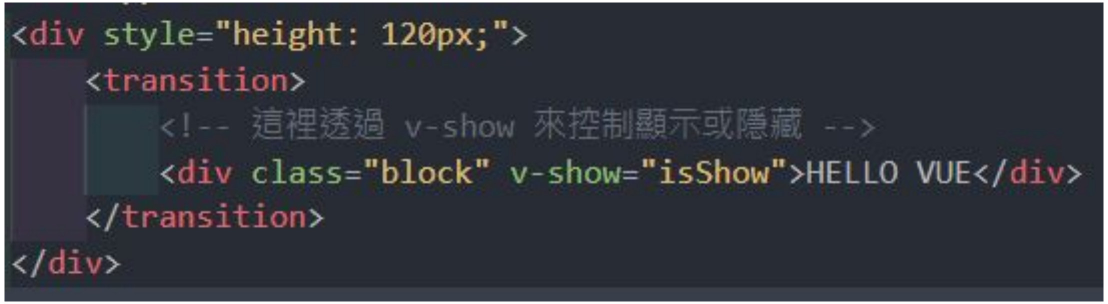
請選擇: 繁體中文

繁體中文 哈囉！世界！

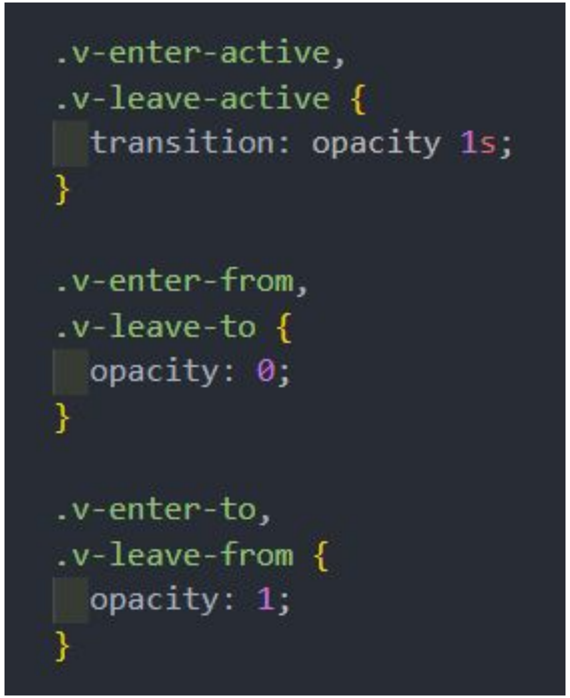
2.5 <transition> 漸變與動畫

2.5.1 <transition> 漸變





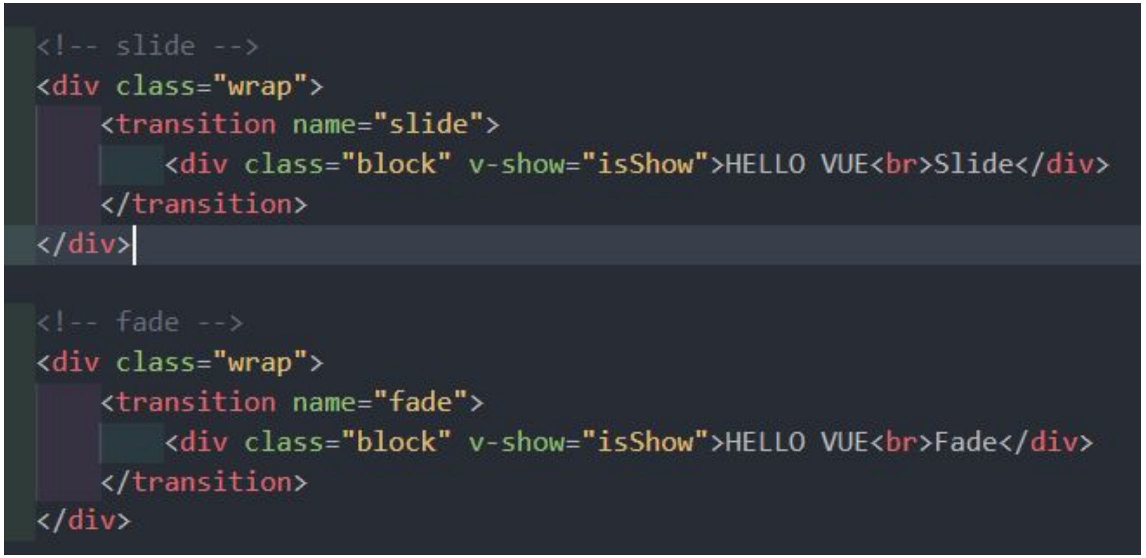
```html
<div style="height: 120px;">
    <transition>
        <!-- 這裡透過 v-show 來控制顯示或隱藏 -->
        <div class="block" v-show="isShow">HELLO VUE</div>
    </transition>
</div>
```



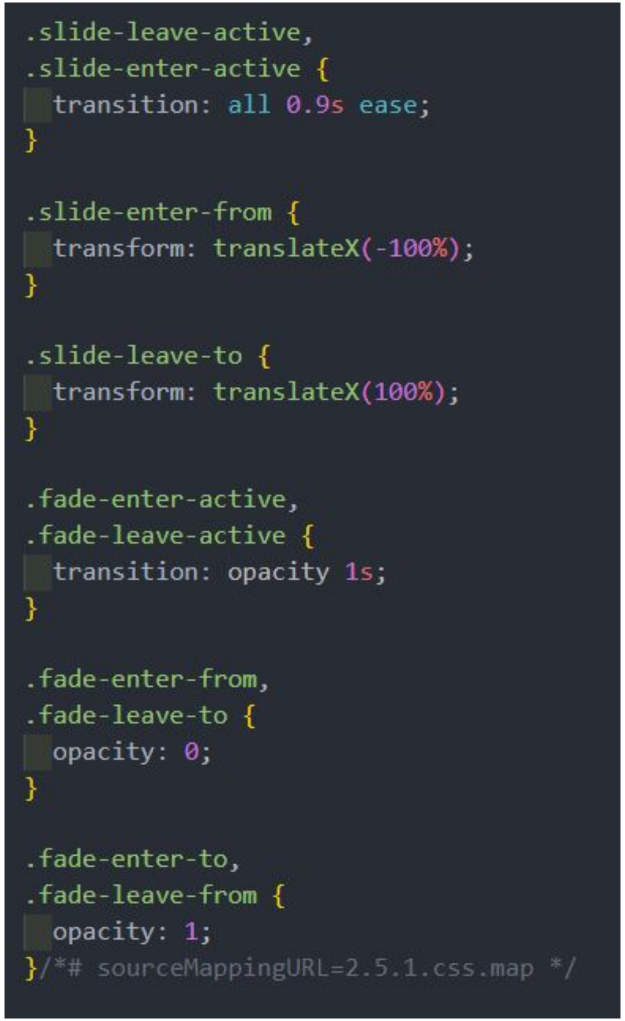
```css
.v-enter-active,
.v-leave-active {
  transition: opacity 1s;
}

.v-enter-from,
.v-leave-to {
  opacity: 0;
}

.v-enter-to,
.v-leave-from {
  opacity: 1;
}
```

Q: 試做一個按鈕, 點擊可讓文字漸變消失。

A: 示範解答

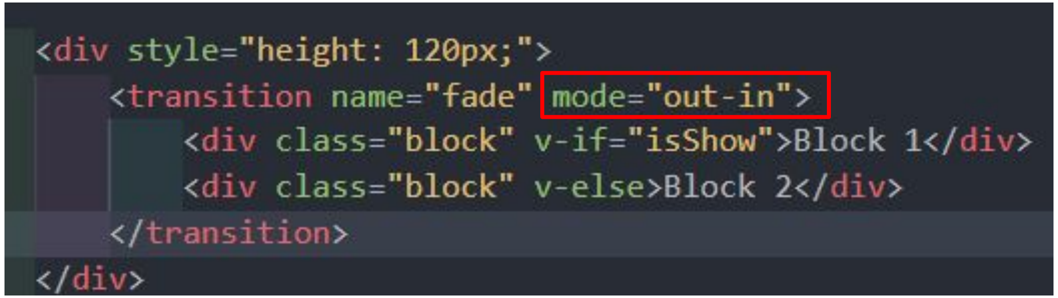


```html
<!-- slide -->
<div class="wrap">
    <transition name="slide">
        <div class="block" v-show="isShow">HELLO VUE<br>Slide</div>
    </transition>
</div>

<!-- fade -->
<div class="wrap">
    <transition name="fade">
        <div class="block" v-show="isShow">HELLO VUE<br>Fade</div>
    </transition>
</div>
```



```css
.slide-leave-active,
.slide-enter-active {
  transition: all 0.9s ease;
}

.slide-enter-from {
  transform: translateX(-100%);
}

.slide-leave-to {
  transform: translateX(100%);
}

.fade-enter-active,
.fade-leave-active {
  transition: opacity 1s;
}

.fade-enter-from,
.fade-leave-to {
  opacity: 0;
}

.fade-enter-to,
.fade-leave-from {
  opacity: 1;
}/*# sourceMappingURL=2.5.1.css.map */
```

2.5.2 條件與動態切換



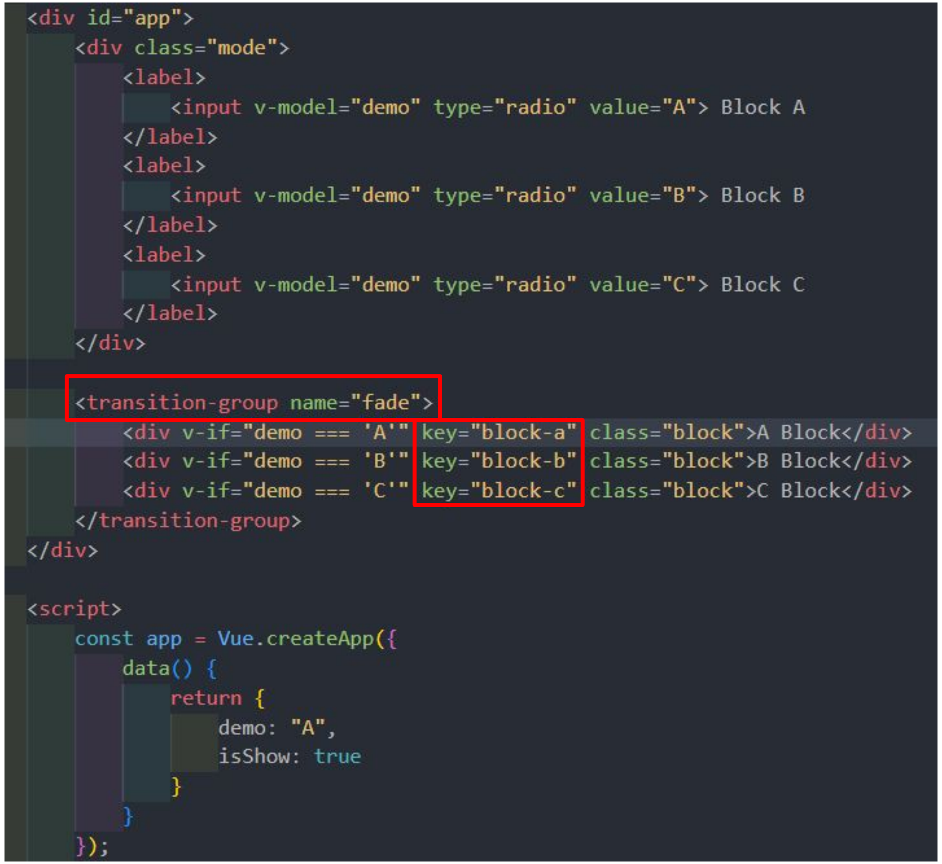
```
<div style="height: 120px;">
    <transition name="fade" mode="out-in">
        <div class="block" v-if="isShow">Block 1</div>
        <div class="block" v-else>Block 2</div>
    </transition>
</div>
```

更改漸變效果的順序

Q : 試做兩個 radio btn, 點擊可更換 mode。

A : 示範解答

2.5.3 複數元素/元件的漸變渲染 <transition-group>



```
<div id="app">
    <div class="mode">
        <label>
            <input v-model="demo" type="radio" value="A"> Block A
        </label>
        <label>
            <input v-model="demo" type="radio" value="B"> Block B
        </label>
        <label>
            <input v-model="demo" type="radio" value="C"> Block C
        </label>
    </div>

    <transition-group name="fade">
        <div v-if="demo === 'A'" key="block-a" class="block">A Block</div>
        <div v-if="demo === 'B'" key="block-b" class="block">B Block</div>
        <div v-if="demo === 'C'" key="block-c" class="block">C Block</div>
    </transition-group>
</div>

<script>
    const app = Vue.createApp({
        data() {
            return {
                demo: "A",
                isShow: true
            }
        }
    });
```

不支援 mode

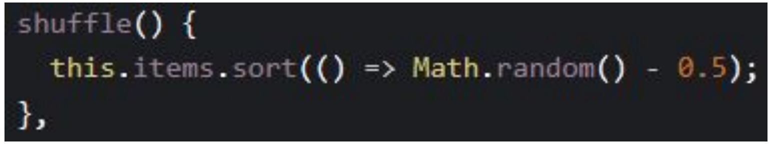
必定加入 key 唯一屬性

實務上通常與 v-for 一起使用

範例

可以為其再增加 shuffle 重新排列

shuffle 方法:



```
shuffle() {
  this.items.sort(() => Math.random() - 0.5);
},
```

2.5.5 結合漸變動畫的 Hooks 函式處理事件