

课程目标



Mixins



HOC



Renderless 组件

Mxins

活动名称

活动区域

请选择活动区域

活动时间

选择日期

-

选择时间

即时配送

活动性质

☐ 美食/餐厅线上活动

☐ 地推活动

☐ 线下主题活动

☐ 单纯品牌曝光

特殊资源

☐ 线上品牌商赞助

☐ 线下场地免费

活动形式

立即创建

取消

封装一个表单组件库。例如各组件都要全部校验才能，提交表单。这时候

Input	Select	Timepicker	Switch	Checkbox
❖ validate	❖ validate	❖ validate	❖ validate	❖ validate

重复的validate校验。这时候需要Mixin混入。Mixin除了全局混入，也可局部混入。

```

1  <template>
2    <div>
3      <input type="text" @blur="blur" />
4      {{ errmsg }}
5    </div>
6  </template>
7
8  <script>
9    import validateMixin from "./mixin";
10
11    export default {
12      mixins: [validateMixin],
13      data: () => ({ errmsg: "" }),
14      methods: {
15        blur() {
16          this.validate();
17        }
18      }
19    };
20  </script>

```

validateMixin

```
1  export default {
2    methods: {
3      validate() {
4        /**
5         * 校验逻辑
6         */
7
8        return true;
9      }
10   }
11 };
12
```

mixin 缺陷 打破了原有组件的封装 增加组件复杂度。可能会出现命名冲突问题，仅仅只对js逻辑复用，模板不能复用。

HOC(higher order component 高阶组件)

定义:函数接收一个组件作为参数,并返回新的组件，可复用的逻辑在函数中实现。其实这个函数相当于一个装饰着函数。

HOC组件

```
1 import Vue from "vue";
2 const ValidateHoc = Component => {
3   return Vue.component(`hoc-${Component.name}`, {
4     data: () => ({ errMsg: "" }),
5     methods: {
6       validate() {
7         // eslint-disable-next-line no-console
8         console.log("validate");
9         /**
10          * 校验逻辑
11          */
12         return true;
13       }
14     },
15     render() {
16       return (
17         <div>
18           <Component on-blur={this.validate} />
19           {this.errMsg}
20         </div>
21       );
22     },
23   });
24 };
25 export default ValidateHoc;
```

input组件

```
1 <template>
2   <input type="text" @blur="$emit('blur')" />
3 </template>
```

外层组件（组装HOC组件和input组件）

```
1 import CustomInput from "../components/composition/2/CustomInput";
2 import ValidateHoc from "../components/composition/2/Hoc.js";
3
4 const ValidateInput = ValidateHoc(CustomInput);
5
6 export default {
7   name: "app",
8   render() {
9     return <ValidateInput />;
10 }
11 };
```

相比较Mixin的优点：

- 模板可复用
- 不会出现命名冲突（本质上是一个HOC是套了一层父组件）

不足：

- 组件复杂度高，多层嵌套，调试会很痛苦

官方也不推荐使用这种方式

Renderless组件

复用的逻辑沉淀在包含slot插槽的组件

接口由插槽Prop来暴露

子组件

```
1 <template>
2   <div>
3     <slot :validate="validate"></slot>
4     {{ errMsg }}
5   </div>
```

```

6 </template>
7 <script>
8 export default {
9   props: ["value", "rules"],
10   data() {
11     return { errMsg: "" };
12   },
13   methods: {
14     validate() {
15       let validate = this.rules.reduce((pre, cur) => {
16         let check = cur && cur.test && cur.test(this.value);
17         this.errMsg = check ? "" : cur.message;
18         return pre && check;
19       }, true);
20       return validate;
21     }
22   }
23 };
24 </script>

```

父组件(接口由插槽prop暴露，复用的逻辑沉淀在包含slot插槽的组件)

```

1 <template>
2   <div>
3     <s-validate #default="{ validate }" :value="value" :rules="rules">
4       <input type="text" @blur="validate" v-model="value" />
5     </s-validate>
6
7     <s-validate #default="{ validate }" :value="text" :rules="textRules">
8       <textarea type="text" @blur="validate" v-model="text" />
9     </s-validate>
10   </div>
11 </template>
12
13 <script>
14 import SValidate from "./SValidate";
15
16 export default {
17   data: () => ({
18     value: "hi",
19     text: "hi",

```

```
20 rules: [  
21   {  
22     test: function(value) {  
23       return /\d+/.test(value);  
24     },  
25     message: "请输入一个数字"  
26   }  
27 ],  
28 textRules: [  
29   {  
30     test: function(value) {  
31       return value;  
32     },  
33     message: "请输入一个非空的值"  
34   }  
35 ]  
36 }},  
37 components: {  
38   SValidate  
39 }  
40 };  
41 </script>
```

我们可以看到子组件，slot流出的插槽，是给父组件插入的，接口由插槽prop暴露（通过prop把 validate给暴露给父组件），下面是可复用的模板{{errMsg}}，也不会造成命名冲突。

优点：

- 模板可复用
- 不会出现命名冲突
- 符合依赖倒置原则
- 复用的接口来源清晰