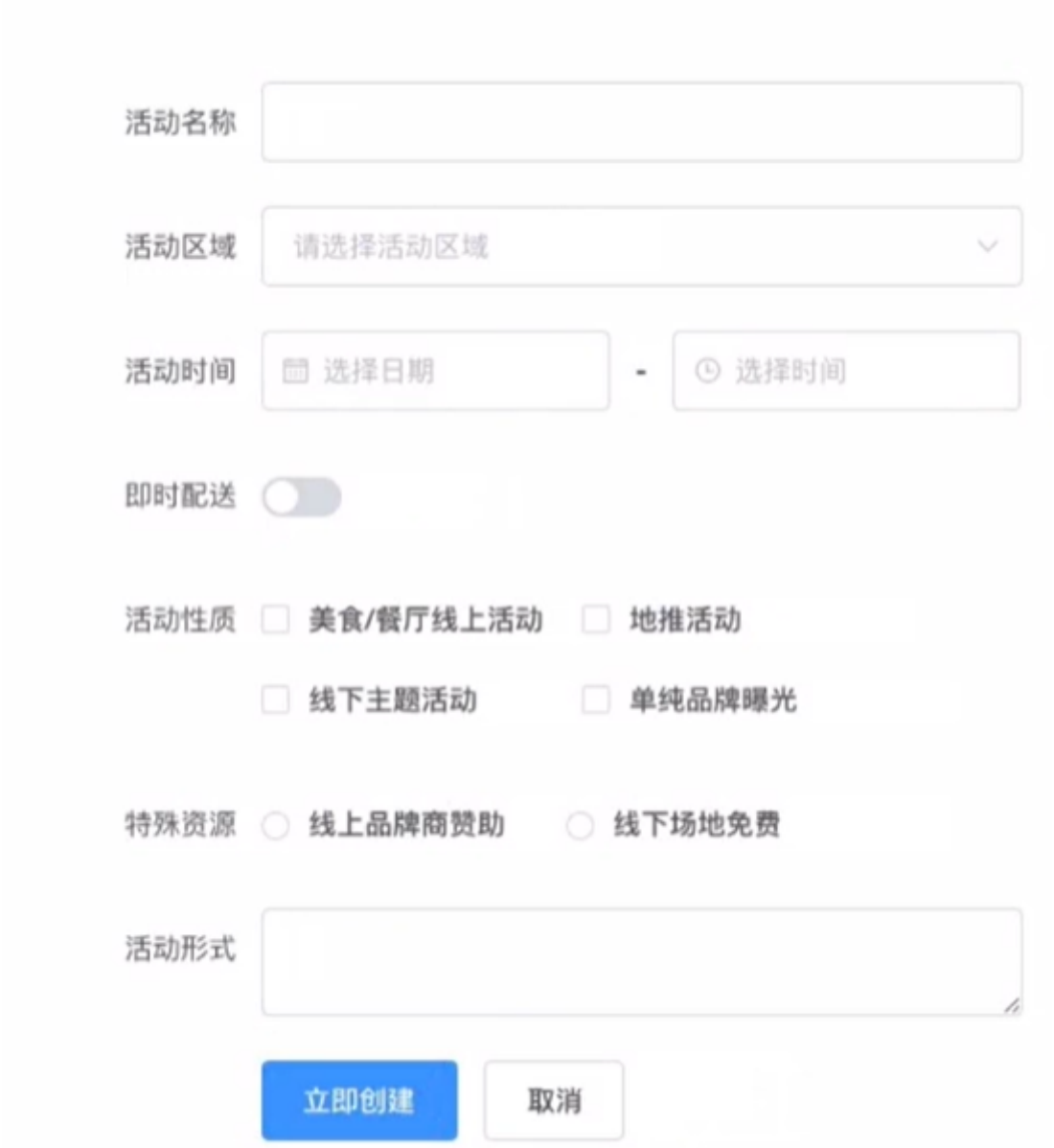
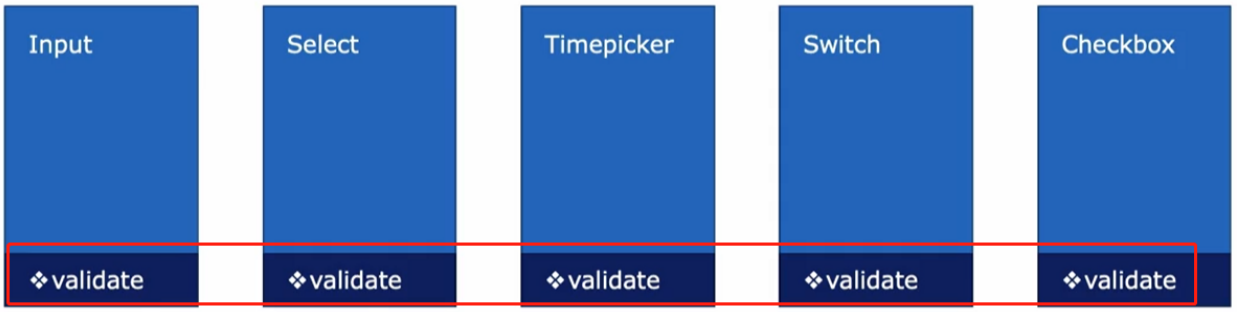
课程目标

Mxins



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



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。

```vue
<template>
  <div>
    <input type="text" @blur="blur" />
    {{ errmsg }}
  </div>
</template>

<script>
import validateMixin from "./mixin";

export default {
  mixins: [validateMixin],
  data: () => ({ errmsg: "" }),
  methods: {
    blur() {
      this.validate();
    }
  }
};
</script>
```

validateMixin

```
export default {
  methods: {
    validate() {
      /**
       * 校验逻辑
       */

      return true;
    }
  }
};
```

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

```
import Vue from "vue";
const ValidateHoc = Component => {
  return Vue.component(`hoc-${Component.name}`, {
    data: () => ({ errMsg: "" }),
    methods: {
      validate() {
        // eslint-disable-next-line no-console
        console.log("validate");
        /**
         * 校验逻辑
         */

        return true;
      }
    },
    render() {
      return (
        <div>
          <Component on-blur={this.validate} />
          {this.errMsg}
        </div>
      );
    }
  });
};
export default ValidateHoc;
```

input组件

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

外层组件（组装HOC组件和input组件）

```
import CustomInput from "./components/composition/2/CustomInput";
import ValidateHoc from "./components/composition/2/Hoc.js";

const ValidateInput = ValidateHoc(CustomInput);

export default {
  name: "app",
  render() {
    return <ValidateInput />;
  }
};
```

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

```
<template>
 <div>
 <slot :validate="validate"></slot>
 {{ errMsg }}
 </div>
```

```
</template>
<script>
export default {
 props: ["value", "rules"],
 data() {
 return { errMsg: "" };
 },
 methods: {
 validate() {
 let validate = this.rules.reduce((pre, cur) => {
 let check = cur && cur.test && cur.test(this.value);
 this.errMsg = check ? "" : cur.message;
 return pre && check;
 }, true);
 return validate;
 }
 }
};
</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)

```
<template>
 <div>
 <s-validate #default="{ validate }" :value="value" :rules="rules">
 <input type="text" @blur="validate" v-model="value" />
 </s-validate>

 <s-validate #default="{ validate }" :value="text" :rules="textRules">
 <textarea type="text" @blur="validate" v-model="text" />
 </s-validate>
 </div>
</template>

<script>
import SValidate from "./SValidate";

export default {
 data: () => ({
 value: "hi",
 text: "hi",
```

```
  rules: [
  {
  test: function(value) {
  return /\d+/.test(value);
  },
  message: "请输入一个数字"
  }
  ],
  textRules: [
  {
  test: function(value) {
  return value;
  },
  message: "请输入一个非空的值"
  }
  ]
  }),
  components: {
  SValidate
  }
};
</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。