



下载APP



22 | 表单：如何设计一个表单组件？

2021-12-10 大圣

《玩转Vue 3全家桶》

课程介绍 >



讲述：大圣

时长 07:27 大小 6.83M



你好，我是大圣。

上一讲我们详细讲解了如何使用 Jest 框架对组件库进行测试，TypeScript 和 Jest 都为我们的代码质量和研发效率保驾护航。之前我们实现的 Container 和 Button 组件都是以渲染功能为主，可以根据不同的属性渲染不同的样式去实现布局和不同格式的按钮。

那么今天我再带你实现一个非常经典的表单组件，这个组件除了要渲染页面组件之外，还得支持很好的页面交互，下面我们先从 Element3 的表单组件开始讲解。



表单组件

在 [Element 表单组件](#) 的页面里，我们能看到表单种类的组件类型有很多，我们常见的输入框、单选框和评分组件等都算是表单组件系列的。

下面这段代码是 Element3 官方演示表单的 Template，整体表单页面分三层：

el-form 组件负责最外层的表单容器；

el-form-item 组件负责每一个输入项的 label 和校验管理；

内部的 el-input 或者 el-switch 负责具体的输入组件。

[复制代码](#)

```
1 <el-form
2   :model="ruleForm"
3   :rules="rules"
4   ref="form"
5   label-width="100px"
6   class="demo-ruleForm"
7 >
8   <el-form-item label="活动名称" prop="name">
9     <el-input v-model="ruleForm.name"></el-input>
10  </el-form-item>
11  <el-form-item label="活动区域" prop="region">
12    <el-select v-model="ruleForm.region" placeholder="请选择活动区域">
13      <el-option label="区域一" value="shanghai"></el-option>
14      <el-option label="区域二" value="beijing"></el-option>
15    </el-select>
16  </el-form-item>
17  <el-form-item label="即时配送" prop="delivery">
18    <el-switch v-model="ruleForm.delivery"></el-switch>
19  </el-form-item>
20  <el-form-item label="活动性质" prop="type">
21    <el-checkbox-group v-model="ruleForm.type">
22      <el-checkbox label="美食/餐厅线上活动" name="type"></el-checkbox>
23      <el-checkbox label="地推活动" name="type"></el-checkbox>
24      <el-checkbox label="线下主题活动" name="type"></el-checkbox>
25      <el-checkbox label="单纯品牌曝光" name="type"></el-checkbox>
26    </el-checkbox-group>
27  </el-form-item>
28  <el-form-item label="特殊资源" prop="resource">
29    <el-radio-group v-model="ruleForm.resource">
30      <el-radio label="线上品牌商赞助"></el-radio>
31      <el-radio label="线下场地免费"></el-radio>
32    </el-radio-group>
33  </el-form-item>
34  <el-form-item label="活动形式" prop="desc">
35    <el-input type="textarea" v-model="ruleForm.desc"></el-input>
36  </el-form-item>
```

```
37   <el-form-item>
38     <el-button type="primary" @click="submitForm('ruleForm')">
39       立即创建</el-button>
40   >
41   <el-button @click="resetForm('ruleForm')">重置</el-button>
42 </el-form-item>
43 </el-form>
```

现在我们把上面的代码简化为最简单的形式，只留下 el-input 作为输入项，就可以清晰地看到表单组件工作的模式：el-form 组件使用:model 提供数据绑定；使用 rules 提供输入校验规则，可以规范用户的输入内容；使用 el-form-item 作为输入项的容器，对输入进行校验，显示错误信息。

[复制代码](#)

```
1   <el-form :model="ruleForm" :rules="rules" ref="form">
2     <el-form-item label="用户名" prop="username">
3       <el-input v-model="ruleForm.username"></el-input>
4       <!-- <el-input :model-value="" @update:model-value=""></el-input> -->
5     </el-form-item>
6     <el-form-item label="密码" prop="passwd">
7       <el-input type="password" v-model="ruleForm.passwd"></el-input>
8     </el-form-item>
9     <el-form-item>
10      <el-button type="primary" @click="submitForm()">登录</el-button>
11    </el-form-item>
12  </el-form>
```

然后我们看下 rules 和 model 是如何工作的。

这里使用 reactive 返回用户输入的数据，username 和 passwd 输入项对应，然后 rules 使用 reactive 包裹用户输入项校验的配置。

具体的校验规则，现在主流组件库使用的都是 async-validator 这个库，详细的校验规则你可以访问 [async-validator 的官网](#) 查看。而表单 Ref 上我们额外新增了一个 validate 方法，这个方法会执行所有的校验逻辑来显示用户的报错信息，下图就是用户输入不符合 rules 配置后，页面的报错提示效果。

[复制代码](#)

```
1  const ruleForm = reactive<UserForm>({
2    username: "",
```

```
3   passwd: ""
4 })
5 const rules = reactive({
6   rules: {
7     username: { required: true, min: 1, max: 20, message: '长度在 1 到 20 个字符'
8     passwd: [{ required: true, message: '密码', trigger: 'blur' }]
9   }
10 })
11 function submitForm() {
12   form.value.validate((valid) => {
13     if (valid) {
14       alert('submit!')
15     } else {
16       console.log('error submit!!')
17       return false
18     }
19   })
20 }
```

* 活动名称

请输入活动名称

* 活动区域

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

请选择活动资源

* 活动形式

请填写活动形式

立即创建

重置

表单组件实现

那么接下来我们就要实现组件了。我们进入到 `src/components` 目录下新建 `Form.vue` 去实现 `el-form` 组件，该组件是整个表单组件的容器，负责管理每一个 `el-form-item` 组件的校验方法，并且自身还提供一个检查所有输入项的 `validate` 方法。

在下面的代码中，我们注册了传递的属性的格式，并且注册了 `validate` 方法使其对外暴露使用。

[复制代码](#)

```
1
2 interface Props {
3   label?: string
4   prop?: string
5 }
6 const props = withDefaults(defineProps<Props>(), {
7   label: "",
8   prop: ""
9 })
10
11 const formData = inject(key)
12
13 const o: FormItem = {
14   validate,
15 }
16
17 defineExpose(o)
```

那么在 `el-form` 组件中如何管理 `el-form-item` 组件呢？我们先要新建 `FormItem.vue` 文件，这个组件加载完毕之后去通知 `el-form` 组件自己加载完毕了，这样在 `el-form` 中我们就可以很方便地使用数组来管理所有内部的 `form-item` 组件。

[复制代码](#)

```
1 import { emitter } from "../../emitter"
2 const items = ref<FormItem[]>([])
3
4 emitter.on("addFormItem", (item) => {
5   items.value.push(item)
6 })
```

然后 `el-form-item` 还要负责管理内部的 `input` 输入标签，并且从 `form` 组件中获得配置的 `rules`，通过 `rules` 的逻辑，来判断用户的输入值是否合法。另外，`el-form` 还要管理当前输入框的 `label`，看看输入状态是否报错，以及报错的信息显示，这是一个承上启下的组件。

[复制代码](#)

```
1 onMounted(() => {
```

```
2   if (props.prop) {
3     emitter.on("validate", () => {
4       validate()
5     })
6     emitter.emit("addFormItem", o)
7   }
8 })
9 function validate() {
10  if (formData?.rules === undefined) {
11    return Promise.resolve({ result: true })
12  }
13  const rules = formData.rules[props.prop]
14  const value = formData.model[props.prop]
15  const schema = new Schema({ [props.prop]: rules })
16  return schema.validate({ [props.prop]: value }, (errors) => {
17    if (errors) {
18      error.value = errors[0].message || "校验错误"
19    } else {
20      error.value = ""
21    }
22  })
23 }
```

这里我们可以看到，form、form-item 和 input 这三个组件之间是**嵌套使用**的关系：

form 提供了所有的数据对象和配置规则；

input 负责具体的输入交互；

form-item 负责中间的数据和规则管理，以及显示具体的报错信息。

这就需要有一个强有力的组件通信机制，在 Vue 中组件之间的通信机制有这么几种。


首先是父子组件通信，通过 props 和 emits 来通信。这个我们在全家桶实战篇和评级组件那一讲都有讲过，父元素通过 props 把需要的数据传递给子元素，子元素通过 emits 通知父元素内部的变化，并且还可以通过 defineDepose 的方式暴露给父元素方法，可以让父元素调用自己的方法。

那么 form 和 input 组件如何通信呢？这种祖先元素和后代元素，中间可能嵌套了很多层的关系，Vue 则提供了 provide 和 inject 两个 API 来实现这个功能。

在组件中我们可以使用 provide 函数向所有子组件提供数据，子组件内部通过 inject 函数注入使用。注意这里 provide 提供的只是普通的数据，并没有做响应式的处理，如果子组


件内部需要响应式的数据，那么需要在 `provide` 函数内部使用 `ref` 或者 `reactive` 包裹才可以。

关于 `provide` 和 `inject` 的类型系统，我们可以使用 Vue 提供的 `InjectionKey` 来声明。我们在 `form` 目录下新建 `type.ts` 专门管理表单组件用到的相关类型，在下面的代码中，我们定义了表单 `form` 和表单管理 `form-item` 的上下文，并且通过 `InjectionKey` 管理提供的类型。

 复制代码

```
1 import { InjectionKey } from "vue"
2 import { Rules, Values } from "async-validator"
3
4 export type FormData = {
5   model: Record<string, unknown>
6   rules?: Rules
7 }
8
9 export type FormItem = {
10   validate: () => Promise<Values>
11 }
12
13 export type FormType = {
14   validate: (cb: (isValid: boolean) => void) => void
15 }
16
17 export const key: InjectionKey<FormData> = Symbol("form-data")
```


而下面的代码，我们则通过 `provide` 向所有子元素提供 `form` 组件的上下文。子组件内部通过 `inject` 获取，很多组件都是嵌套成对出现的，`provide` 和 `inject` 这种通信机制后面我们还会不停地用到，做好准备。

 复制代码

```
1 provide(key, {
2   model: props.model,
3   rules?: props.rules,
4 })
5
6 # 子组件
7 const formData = inject(key)
```

然后就是具体的 input 实现逻辑, 在下面的代码中, input 的核心逻辑就是对 v-model 的支持, 这个内容我们在评级组件那一讲已经实现过了。

v-model 其实是: mode-value="x" 和 @update:modelValue 两个写法的简写, 组件内部获取对应的属性和 modelValue 方法即可。这里需要关注的代码是我们输入完成之后的事件, 输入的结果校验是由父组件 el-form-item 来实现的, 我们只需要通过 emit 对外广播出去即可。

 复制代码

```
1 <template>
2   <div
3     class="el-form-item"
4   >
5     <label
6       v-if="label"
7     >{{ label }}</label>
8     <slot />
9     <p
10      v-if="error"
11      class="error"
12    >
13      {{ error }}
14    </p>
15  </div>
16 </template>
17 <script lang="ts">
18 export default{
19   name: 'ElFormItem'
20 }
21 </script>
22
23 <script setup lang="ts">
24 import Schema from "async-validator"
25 import { onMounted, ref, inject } from "vue"
26 import { FormItem, key } from "../type"
27 import { emitter } from "../../emitter"
28
29 interface Props {
30   label?: string
31   prop?: string
32 }
33 const props = withDefaults(defineProps<Props>(), { label: "", prop: "" })
34 // 错误
35 const error = ref("")
36
37 const formData = inject(key)
38
```



```
39  const o: FormItem = {
40    validate,
41  }
42
43  defineExpose(o)
44
45  onMounted(() => {
46    if (props.prop) {
47      emitter.on("validate", () => {
48        validate()
49      })
50      emitter.emit("addFormItem", o)
51    }
52  })
53
54  function validate() {
55    if (formData?.rules === undefined) {
56      return Promise.resolve({ result: true })
57    }
58    const rules = formData.rules[props.prop]
59    const value = formData.model[props.prop]
60    const schema = new Schema({ [props.prop]: rules })
61    return schema.validate({ [props.prop]: value }, (errors) => {
62      if (errors) {
63        error.value = errors[0].message || "校验错误"
64      } else {
65        error.value = ""
66      }
67    })
68  }
69 </script>
70
71 <style lang="scss">
72 @import '../styles/mixin';
73 @include b(form-item) {
74   margin-bottom: 22px;
75   label{
76     line-height: 1.2;
77     margin-bottom: 5px;
78     display: inline-block;
79   }
80   & .el-form-item {
81     margin-bottom: 0;
82   }
83 }
84 .error{
85   color: red;
86 }
87 </style>
```

最后我们点击按钮的时候，在最外层的 form 标签内部会对所有的输入项进行校验。由于我们管理着所有的 form-item，只需要遍历所有的 form-item，依次执行即可。

下面的代码就是表单注册的 validate 方法，我们遍历全部的表单输入项，调用表单输入项的 validate 方法，有任何一个输入项有报错信息，整体的校验就会是失败状态。

[复制代码](#)

```
1
2
3 function validate(cb: (isValid: boolean) => void) {
4   const tasks = items.value.map((item) => item.validate())
5   Promise.all(tasks)
6     .then(() => { cb(true) })
7     .catch(() => { cb(false) })
8 }
```

上面代码实际执行的是每个表单输入项内部的 validate 方法，这里我们使用的就是 async-validate 的校验函数。在 validate 函数内部，我们会获取表单所有的 rules，并且过滤出当前输入项匹配的输入校验规则，然后通过 AsyncValidator 对输入项进行校验，把所有的校验结果放在 model 对象中。如果 errors[0].message 非空，就说明校验失败，需要显示对应的错误消息，页面输入框显示红色状态。

[复制代码](#)

```
1 import Schema from "async-validator"
2
3 function validate() {
4   if (formData?.rules === undefined) {
5     return Promise.resolve({ result: true })
6   }
7   const rules = formData.rules[props.prop]
8   const value = formData.model[props.prop]
9   const schema = new Schema({ [props.prop]: rules })
10  return schema.validate({ [props.prop]: value }, (errors) => {
11    if (errors) {
12      error.value = errors[0].message || "校验错误"
13    } else {
14      error.value = ""
15    }
16  })
17 }
```

总结

今天的课程到这就结束了，我们来总结一下今天学到的内容吧。

今天我们设计和实现了一个比较复杂的组件类型——表单组件。表单组件在组件库中作用，就是收集和获取用户的输入值，并且提供用户的输入校验，比如输入的长度、邮箱格式等，符合校验规则后，就可以获取用户输入的内容，并提交给后端。

这一过程中我们要实现三类组件：

el-form 提供表单的容器组件，负责全局的输入对象 model 和校验规则 rules 的配置，并且在用户点击提交的时候，可以执行全部输入项的校验规则；

其次是 input 类组件，我们日常输入内容的输入框、下拉框、滑块等都属于这一类组件，这类组件主要负责显示对应的交互组件，并且监听所有的输入项，用户在交互的同时通知执行校验；

然后就是介于 form 和 input 中间的 form-item 组件，这个组件负责每一个具体输入的管理，从 form 组件中获取校验规则，从 input 中获取用户输入的内容，通过 async-validator 校验输入是否合法后显示对应的输入状态，并且还能把校验方法提供给 form 组件，form 可以很方便地管理所有 form-item。

至此，form 组件设计完毕，相信你对组件通信、输入类组件的实现已经得心应手了，并且对组件设计中如何使用 TypeScript 也有了自己的心得。**组件设计我们需要考虑的就是内部交互的逻辑，对子组件提供什么数据，对父组件提供什么方法，需不需要通过 provide 或者 inject 来进行跨组件通信等等。**相信实践过后，你会有更加深刻的理解和认识。

思考题

最后留一道思考题：今天的表单组件在设计上能否通过 Vue 2 时代流行的 event-bus 来实现呢？

期待在评论区看到你的思考，也欢迎你把这一讲分享给你的同事和朋友们，我们下一讲再见！

[生成海报并分享](#)[赞 4](#)[提建议](#)

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 21 | 单元测试：如何使用 TDD 开发一个组件？

更多课程推荐

跟月影学可视化

系统掌握图形学与可视化核心原理

月影

奇虎 360 奇舞团团长

可视化 UI 框架 SpriteJS 核心开发者



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

精选留言 (3)

[写留言](#)

Johnson

2021-12-10

思考题:可以实现，但是不太优雅！

展开 ∨





海阔天空

2021-12-10

表单组件在设计在element中实现原理是通过 Vue 2 时代流行的 event-bus 来实现的，不过他们自己封装了emit和watch方法。vue3这实现更方便更简单了啊。

作者回复：其实复杂的也得自己封装一下 只不过vue3没有内置了







pzz
2021-12-10

第一？

展开 ▾



