



下载APP



20 | 组件库：如何设计你自己的通用组件库？

2021-12-06 大圣

《玩转Vue 3全家桶》

课程介绍 >

**讲述：大圣**

时长 09:03 大小 8.29M



你好，我是大圣。上一讲 TypeScript 加餐学完，你是不是想赶紧巩固一下 TypeScript 在 Vue 中的使用呢？那么从今天开始，我们就重点攻克 Vue 中组件库的实现难点，我会用 7 讲的篇幅带你进入组件库的世界。

学习路径大致是这样的，首先我会给你拆解一下 Element3 组件库的代码，其次带你剖析组件库中一些经典的组件，比如表单、表格、弹窗等组件的实现细节，整体使用 Vite+TypeScript+Sass 的技术栈来实现。而业务中繁多的页面也是由一个个组件拼接而成的，所以我们可以先学习一下不同类型的组件是如何去设计的，借此举一反三。



环境搭建

下面我们直奔主题，开始搭建环境。这个章节的代码我已经推送到了 [Github](#) 上，由于组件库是模仿 Element 实现的，所以我为其取名为 ailemente。

接下来我们就一步步实现这个组件库吧。首先和开发项目一样，我们要在命令行里使用下面的命令创建 Vite 项目，模板选择 vue-ts，这样我们就拥有了一个 Vite+TypeScript 的开发环境。

```
1 npm init vite@latest
```

[复制代码](#)

```
→ ~ npm init vite@latest
npx: 6 安装成功，用时 2.104 秒
✓ Project name: ... ailemente
✓ Select a framework: > vue
✓ Select a variant: > vue-ts

Scaffolding project in /Users/woniuppp/ailemente...

Done. Now run:

  cd ailemente
  npm install
  npm run dev
```


关于 ESLint 和 Sass 的相关配置，全家桶实战篇我们已经详细配置了，这里只补充一下 husky 的内容。husky 这个库可以很方便地帮助我们设置 Git 的钩子函数，可以允许我们在代码提交之前进行代码质量的监测。

下面的代码中，我们首先安装和初始化了 husky，然后我们使用 `npx husky add` 命令新增了 `commit-msg` 钩子，husky 会在我们执行 `git commit` 提交代码的时候执行 `node scripts/verifyCommit` 命令来校验 `commit` 信息格式。

```
1 npm install -D husky # 安装husky
2 npx husky install    # 初始化husky
3 # 新增commit msg钩子
4 npx husky add .husky/commit-msg "node scripts/verifyCommit.js"
```

[复制代码](#)

然后我们来到项目目录下的 `verifyCommit` 文件。在下面的代码中，我们先去 `.git/COMMIT_EDITMSG` 文件中读取了 `commit` 提交的信息，然后使用了正则去校验提交信息的格式。如果 `commit` 的信息不符合要求，会直接报错并且终止代码的提交。

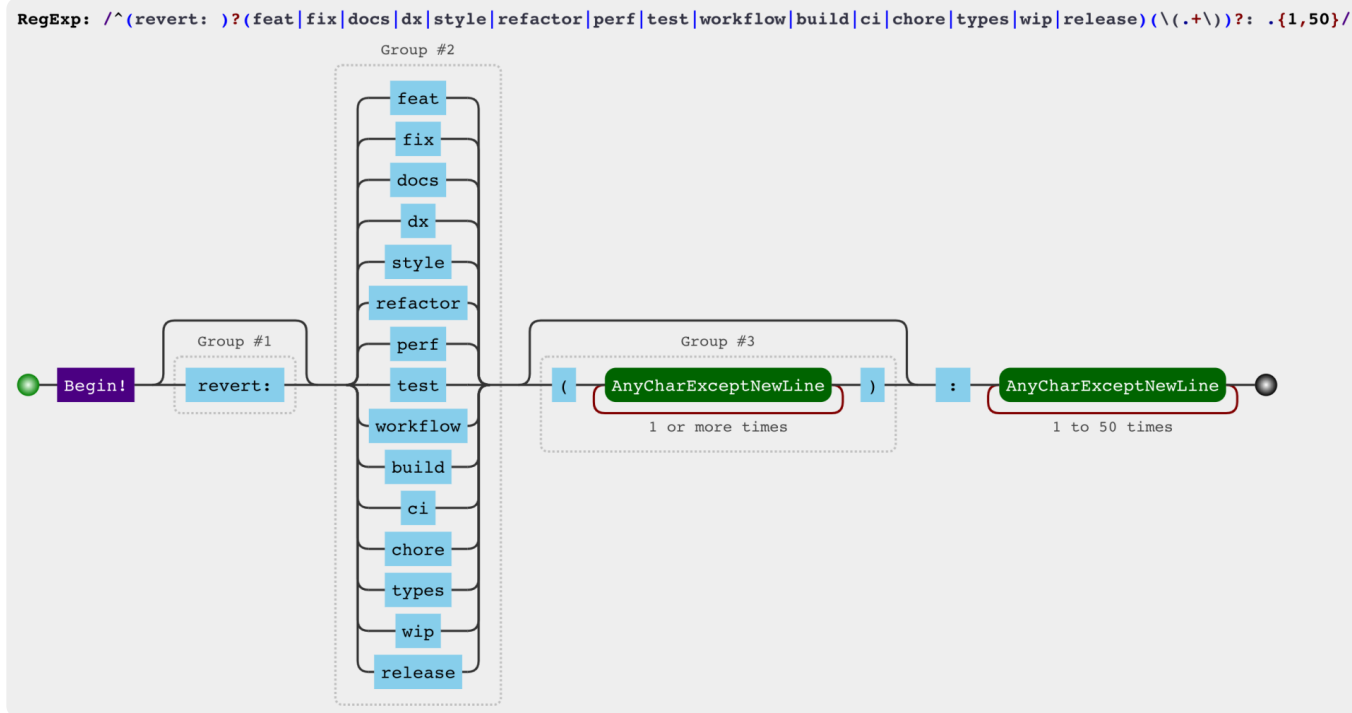
 复制代码

```
1
2
3 const msg = require('fs')
4   .readFileSync('.git/COMMIT_EDITMSG', 'utf-8')
5   .trim()
6
7 const commitRE = /^(revert: )?(feat|fix|docs|dx|style|refactor|perf|test|workf
8 const mergeRe = /^(Merge pull request|Merge branch)/
9 if (!commitRE.test(msg)) {
10   if(!mergeRe.test(msg)){
11     console.log('git commit信息校验不通过')
12
13     console.error(`git commit的信息格式不对，需要使用 title(scope): desc的格式
14       比如 fix: xxbug
15       feat(test): add new
16       具体校验逻辑看 scripts/verifyCommit.js
17     `)
18     process.exit(1)
19   }
20
21 }else{
22   console.log('git commit信息校验通过')
23 }
24
```

这样就确保在 GitHub 中的提交日志都符合 `type(scope): message` 的格式。你可以看下 Vue 3 的 [代码提交记录](#)，每个提交涉及的模块，类型和信息都清晰可见，能够很好地帮助我们管理版本日志，校验正则的逻辑。如下图，`feat` 代表新功能，`docs` 代表文档，`perf` 代表性能。下面的提交日志就能告诉我们这次提交的是组件相关的新功能，代码中新增了 `Button.vue`。

 复制代码

```
1 feat(component): add Button.vue
```



commit-msg 是代码执行提交的时候执行的，我们还可以使用代码执行之前的钩子 pre-commit 去执行 ESLint 代码格式。这样我们在执行 git commit 的同时，就会首先进行 ESLint 校验，然后执行 commit 的 log 信息格式检查，全部通过后代码才能提交至 Git，这也是现在业界通用的解决方案，学完你就快去优化一下手里的项目吧！

[复制代码](#)

```
1 npx husky add .husky/commit-msg "npm run lint"
```

布局组件

好，现在环境我们就搭建好了，接着看看怎么布局组件。

我们可以参考 [Element3 组件列表页面](#)，这里的组件分成了基础组件、表单组件、数据组件、通知组件、导航组件和其他组件几个类型，这些类型基本覆盖了组件库的适用场景，项目中的业务组件也是由这些类型组件拼接而来的。

我们还可以参考项目模块的规范搭建组件库的模板，包括 Sass、ESLint 等，组件库会在这些规范之上加入单元测试来进一步确保代码的可维护性。

接下来我们逐一讲解下各个组件的负责范围。

首先我们需要设计基础的组件，也就是整个项目中都会用到的组件规范，包括布局、色彩，字体、图标等等。这些组件基本没有 JavaScript 的参与，实现起来也很简单，负责的就是项目整体的布局和色彩设计。

而表单组件则负责用户的输入数据管理，包括我们常见的输入框、滑块、评分等等，总结来说，**需要用户输入的地方就是表单组件的应用场景**，其中对用户的输入校验是比较重要的功能点。

数据组件负责显示后台的数据，最重要的就是表格和树形组件。

通知组件负责通知用户操作的状态，包括警告和弹窗，如何用函数动态渲染组件是警告组件的重要功能点。

接下来我们就动手设计一个基础的布局组件，这个组件相对是比较简单的。你可以访问 [🔗 Element3 布局容器页面](#)，这里一共有 container、header、footer、aside、main 五个组件，这个组合可以很方便地实现常见的页面布局。

el-container 组件负责外层容器，当子元素中包含 `<el-header>` 或 `<el-footer>` 时，全部子元素会垂直上下排列，否则会水平左右排列。

el-header、el-aside、el-main、el-footer 组件分别负责顶部和侧边栏，页面主体和底部容器组件。这个功能比较简单，只是渲染页面的布局。我们可以在 `src/components` 目录下新建文件夹 `container`，新建 `Container.vue`，布局组件没有交互逻辑，只需要通过 flex 布局就可以实现。

这几个组件只是提供了不同的 class，这里就涉及到 CSS 的设计内容。在 Element3 中所有的样式前缀都是 el 开头，每次都重复书写维护太困难，所以我们设计之初就需要涉及 Sass 的 Mixin 来提高书写 CSS 的代码效率。

接着，我们在 `src/styles` 下面新建 `mixin.scss`。在下面的代码中，我们定义了 namespace 变量为 el，使用 Mixin 注册一个可以重复使用的模块 b，`b` 可以通过传进来的 block 生成新的变量 `$B`，并且变量会渲染在 class 上，并且注册了 when 可以新增 class 选择器，实现多个 class 的样式。

```

1 // bem
2
3
4 $namespace: 'el';
5 @mixin b($block) {
6   $B: $namespace + '-' + $block !global;
7   .#{ $B } {
8     @content;
9   }
10 }
11
12 // 添加bem后缀啥的
13 @mixin when($state) {
14   @at-root {
15     &.#{$state-prefix + $state} {
16       @content;
17     }
18   }
19 }

```

代码看着有些抽象，不要急，我们再在 container.vue 中写上下面的代码。使用 @import 导入 mixin.scss 后，就可以用 include 语法去使用 Mixin 注册的代码块。

[复制代码](#)

```

1 <style lang="scss">
2 @import '../styles/mixin';
3 @include b(container) {
4   display: flex;
5   flex-direction: row;
6   flex: 1;
7   flex-basis: auto;
8   box-sizing: border-box;
9   min-width: 0;
10  @include when(vertical) {
11    flex-direction: column;
12  }
13 }
14
15 </style>

```

在上面的代码中，我们使用 b(container) 生成 el-container 样式类，在内部使用 when(vertical) 生成 el-container.is-vertical 样式类，去修改 flex 的布局方向。

[复制代码](#)

```

1 .el-container {
2   display: flex;

```

```
3   flex-direction: row;
4   flex: 1;
5   flex-basis: auto;
6   box-sizing: border-box;
7   min-width: 0;
8 }
9 .el-container.is-vertical {
10  flex-direction: column;
11 }
```

container 组件如果内部没有 header 或者 footer 组件，就是横向布局，否则就是垂直布局。根据上面的 CSS 代码，我们可以知道，只需要新增 is-vertical 这个 class，就可以实现垂直布局。

我们在 Container.vue 中写下下面的代码，template 中使用 el-container 容器包裹，通过:class 来实现样式控制即可。然后你肯定会疑惑，为什么会有两个 script 标签？

因为开发组件库的时候，我们要确保每个组件都有自己的名字，script setup 中没法返回组件的名字，所以我们需要一个单独的标签，使用 options 的语法设置组件的 name 属性。

然后在 <script setup> 标签中，添加 lang="ts" 来声明语言是 TypeScript。Typescript 实现组件的时候，我们只需要使用 interface 去定义传递的属性类型即可。使用 defineProps() 实现参数类型校验后，我们再使用 computed 去判断 container 的方向是否为垂直，手动指定 direction 和子元素中有 el-header 或者 el-footer 的时候是垂直布局，其他情况是水平布局。

[复制代码](#)

```
1
2 <template>
3   <section
4     class="el-container"
5     :class="{ 'is-vertical': isVertical }"
6   >
7     <slot />
8   </section>
9 </template>
10 <script lang="ts">
11 export default{
12   name: 'ElContainer'
13 }
```



```
14 </script>
15 <script setup lang="ts">
16
17 import {useSlots,computed,VNode,Component} from 'vue'
18
19 interface Props {
20   direction?:string
21 }
22 const props = defineProps<Props>()
23
24 const slots = useSlots()
25
26 const isVertical = computed(() => {
27   if (slots && slots.default) {
28     return slots.default().some((vn:VNode) => {
29       const tag = (vn.type as Component).name
30       return tag === 'ElHeader' || tag === 'ElFooter'
31     })
32   } else {
33     if (props.direction === 'vertical') {
34       return true
35     } else {
36       return false
37     }
38   }
39 })
40 </script>
```

这样我们的 container 组件就实现了，其他四个组件实现起来大同小异。我们以 header 组件举例，在 container 目录下新建 Header.vue。

在下面的代码中，template 中渲染 el-header 样式类，通过 defineProps 定义传入的属性 height，并且通过 withDefaults 设置 height 的默认值为 60px，通过 b(header) 的方式实现样式，用到的变量都在 style/mixin 中注册，方便多个组件之间的变量共享。

[复制代码](#)

```
1 <template>
2   <header
3     class="el-header"
4     :style="{ height }"
5   >
6     <slot />
7   </header>
8 </template>
9
10 <script lang="ts">
11 export default{
```



```
12   name: 'ElHeader'
13 }
14 </script>
15 <script setup lang="ts">
16 import {withDefaults} from 'vue'
17
18 interface Props {
19   height?:string
20 }
21 withDefaults(defineProps<Props>()),{
22   height:"60px"
23 })
24
25 </script>
26
27 <style lang="scss">
28 @import '../styles/mixin';
29 @include b(header) {
30   padding: $--header-padding;
31   box-sizing: border-box;
32   flex-shrink: 0;
33 }
34
35 </style>
```

组件注册

aside、footer 和 main 组件代码和 header 组件基本一致，你可以在 [这次提交中](#) 看到组件的变更。

组件注册完毕之后，我们在 src/App.vue 中使用 import 语法导入后就可以直接使用了。但是这里有一个小问题，我们的组件库最后会有很多组件对外暴露，用户每次都 import 的话确实太辛苦了，所以我们还需要使用插件机制对外暴露安转的接口，我们在 container 目录下新建 index.ts。在下面的代码中，我们对外暴露了一个对象，对象的 install 方法中，我们使用 ap.component 注册这五个组件。

[复制代码](#)

```
1 import {App} from 'vue'
2 import ElContainer from './Container.vue'
3 import ElHeader from './Header.vue'
4 import ElFooter from './Footer.vue'
5 import ElAside from './Aside.vue'
6 import ElMain from './Main.vue'
7
8 export default {
```

```
9   install(app:App) {
10     app.component(ElContainer.name, ElContainer)
11     app.component(ElHeader.name, ElHeader)
12     app.component(ElFooter.name, ElFooter)
13     app.component(ElAside.name, ElAside)
14     app.component(ElMain.name, ElMain)
15   }
16 }
```

然后我们来到 `src/main.ts` 文件中，下面的代码中我们使用 `app.use(ElContainer)` 的方式注册全部布局组件，这样在项目内部就可以全局使用 `contain`、`header` 等五个组件。实际的组件库开发过程中，[每个组件都会提供一个 `install` 方法](#)，可以很方便地根据项目的需求按需加载。

[复制代码](#)

```
1 import { createApp } from 'vue'
2 import App from './App.vue'
3 import ElContainer from './components/container'
4
5 const app = createApp(App)
6 app.use(ElContainer)
7   .use(ElButton)
8   .mount('#app')
9
```

组件使用

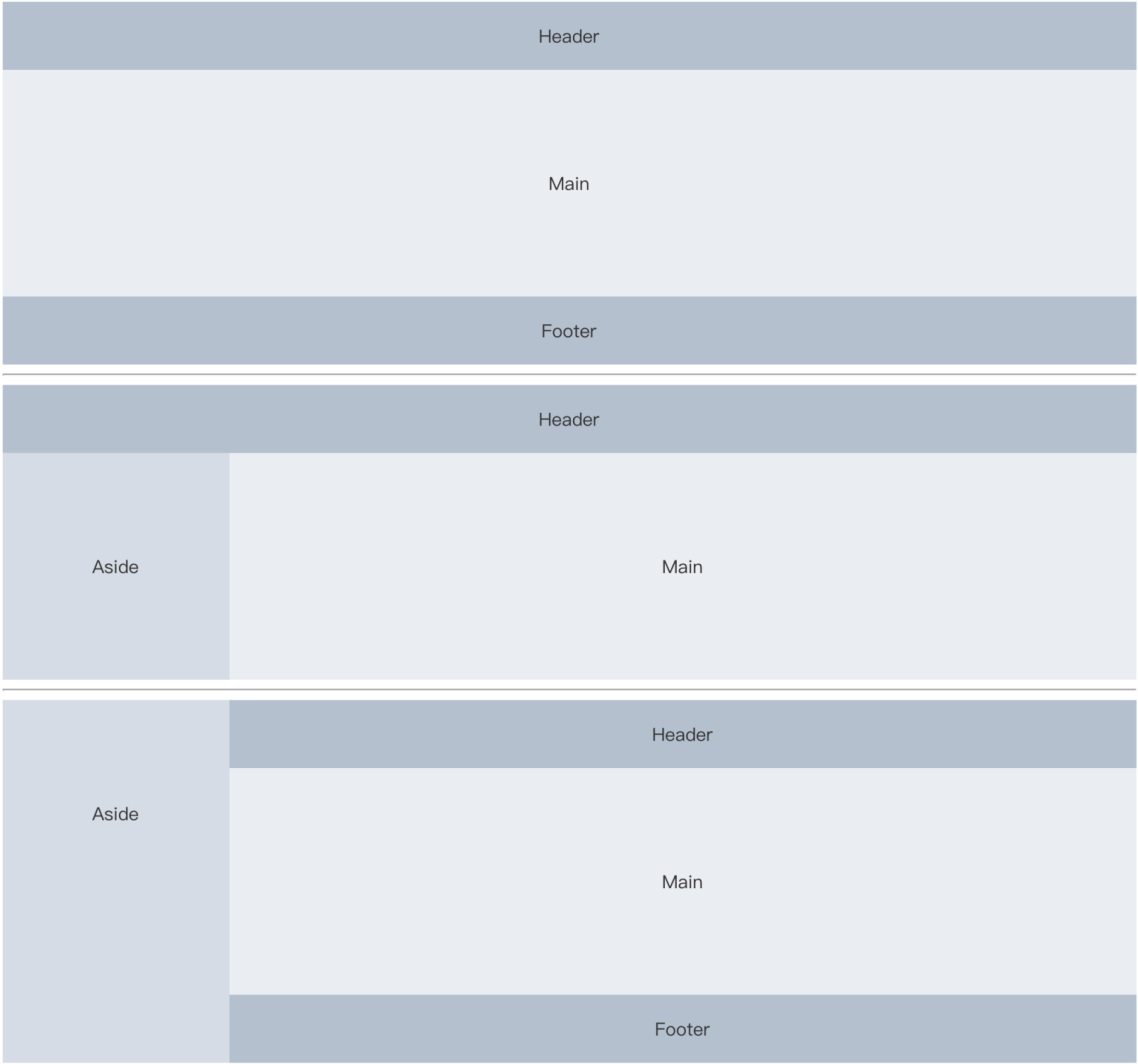
后面我会教你设计组件库的文档系统，这里我们就先用 `App.vue` 来演示一下组件效果。

在 `src/App.vue` 的代码中，我们使用组件嵌套的方式就可以实现下面的页面布局。

[复制代码](#)

```
1 <template>
2   <el-container>
3     <el-header>Header</el-header>
4     <el-main>Main</el-main>
5     <el-footer>Footer</el-footer>
6   </el-container>
7   <hr>
8
9   <el-container>
10    <el-header>Header</el-header>
```

```
11   <el-container>
12     <el-aside width="200px">Aside</el-aside>
13     <el-main>Main</el-main>
14   </el-container>
15 </el-container>
16   <hr>
17 <el-container>
18   <el-aside width="200px">Aside</el-aside>
19   <el-container>
20     <el-header>Header</el-header>
21     <el-main>Main</el-main>
22     <el-footer>Footer</el-footer>
23   </el-container>
24 </el-container>
25 </template>
26 <script setup lang="ts">
27 </script>
28 <style>
29
30 body{
31   width:1000px;
32   margin:10px auto;
33 }
34 .el-header,
35 .el-footer {
36   ..设置额外的css样式
37 }
38 </style>
```



总结

今天的课程到这里就结束了，我们来总结一下今天学到的内容吧。

首先我们介绍了组件库中的组件分类，基础组件、表单组件、数据组件和通知组件，这些组件类型组合拼接，就可以开发出功能繁多的项目。其中，基础组件负责整体布局和色彩的显示，表单组件负责用户的输入，数据组件负责数据的显示，通知组件用于数据的反馈。

然后我们基于 Vite 和 TypeScript 搭建了组件库的代码环境，使用 husky 实现了代码提交之前的 commit 信息和 ESLint 格式校验，确保只有合格的代码能够提交成功。

最后我们使用 TypeScript+Sass 开发了布局相关的组件，container 负责容器，header、footer、aside 和 main 负责渲染不同模块。

这节课我们学会了如何使用 Sass 来提高管理 CSS 代码的效率，以及如何使用 TypeScript 来开发组件，现在相信你对 Vue 3 中如何使用 TypeScript 已经有了更进一步的认识。

思考题

最后我们留个思考题：你负责的业务里有什么组件适合放在组件库里维护呢？

欢迎在评论区分享你的答案，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！

分享给需要的人，Ta订阅后你可得 **20 元现金奖励**

 生成海报并分享

 赞 8  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 [加餐02 | 深入TypeScript](#)

下一篇 [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 (9)

[写留言](#)

.K

2021-12-08

mixin.scss 里少一个变量
\$state-prefix: 'is-';



👍 3



cwang

2021-12-10

大圣老师好，尽管如此文档资源丰富的今天，还是想问你这个问题：

我们安装eslint，还是husky，都是使用npm。但是到了初始化这些安装包的时候，我们却用了npx来做这些，请问这是为什么呢？谢谢。

展开 ▾



全

2021-12-09

feat 代表新功能，docs 代表文档，perf 代表性能 那剩下的表示什么，哪里有注释吗

**bb**

2021-12-07

有没有一个标准的语法模型，我现在都不知道用那种方式了，一下有<script>，<script set up>，setup()。不知道用那种

展开 ∨

作者回复：建议直接<script setup> 如果用option api 就用<script>

**Geek_c7acd2**

2021-12-07

终于看到了想看到的 哈哈

展开 ∨

**szpg**

2021-12-06

npx husky add .husky/commit-msg "node scripts/verifyCommit.js"

不行的，可把 npx 改为 yarn

展开 ∨

共 1 条评论 >

**南山**

2021-12-06

干货满满

展开 ∨

**Danny**

2021-12-06



展开 ∨

**海阔天空**

2021-12-06

老师的这个分类很详细，组件库中的组件分类，基础组件、表单组件、数据组件和通知组件。一般基础组件，表单组件，和通知组件都做成公共组件来进行维护

作者回复：这个其实就是公用组件库的分类

