



下载APP



26 | 文档：如何给你的组件库设计一个可交互式文档？

2021-12-20 大圣

《玩转Vue 3全家桶》

课程介绍 >

**讲述：大圣**

时长 06:12 大小 5.69M



你好，我是大圣。

在我们实现了组件库核心的组件内容之后，我们就需要提供一个可交互的组件文档给用户使用和学习了。这个文档页面主要包含组件的描述，组件 Demo 示例的展示、描述和代码，并且每个组件都应该有详细的参数文档。

现在，我们从上述文档页包含的信息来梳理一下我们的需求。我们需要用最简洁的语法来写页面，还需要用最简洁的语法来展示 Demo + 源代码 + 示例描述。那么从语法上来说，首选就是 Markdown 无疑了，因为它既简洁又强大。



那在我们正式开始设计文档之前，我们还需要对齐一下。如果要展示 Demo 和源码的话，为了能更高效且低成本的维护，我们会把一个示例的 Demo + 源码 + 示例描述放到一个

文件里，尽量多的去复用，这样可以减少需要维护的代码。而做示例展示的话，本质上可以说是跟 Markdown 的转译一致，都是 Markdown -> HTML，只是转译的规则我们需要拓展一下。接下来我们就正式开始了。

VuePress

首先我们需要一个能基于 Markdown 构建文档的工具，我推荐 VuePress。它是 Vue 官网团队维护的在线技术文档工具，样式和 Vue 的官方文档保持一致。

VuePress 内置了 Markdown 的扩展，写文档的时候就是用 Markdown 语法进行渲染的。最让人省心的是，它可以直接在 Markdown 里面使用 Vue 组件，这就意味着我们可以直接在 Markdown 中写上一个个的组件库的使用代码，就可以直接展示运行效果了。

我们可以在项目中执行下面的代码安装 VuePress 的最新版本：

```
1 yarn add -D vuepress@next
```

[复制代码](#)

然后我们新建 docs 目录作为文档目录，新建 docs/README.md 文件作为文档的首页。除了 Markdown 之外，我们可以直接使用 VuePress 的语法扩展对组件进行渲染。

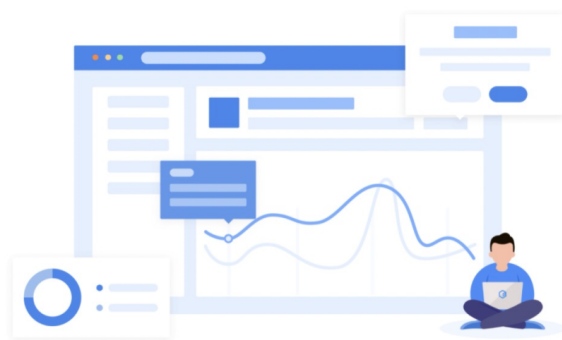
```
1
2 ---
3 home: true
4 heroImage: /theme.png
5 title: 网站快速成型工具
6 tagline: 一套为开发者、设计师和产品经理准备的基于 Vue 3 的桌面端组件库
7 heroText: 网站快速成型工具
8 actions:
9   - text: 快速上手
10     link: /install
11     type: primary
12   - text: 项目简介
13     link: /button
14     type: secondary
15 features:
16   - title: 简洁至上
17     details: 以 Markdown 为中心的项目结构，以最少的配置帮助你专注于写作。
18   - title: Vue 驱动
19     details: 享受 Vue 的开发体验，可以在 Markdown 中使用 Vue 组件，又可以使用 Vue 来开
```

[复制代码](#)

```
20   - title: 高性能
21     details: VuePress 会为每个页面预渲染生成静态的 HTML，同时，每个页面被加载的时候，将
22 footer: powdered by vuepress and me
23 ---
24 # 额外的信息
25
26
27
```

我们在 README.md 中输入上面的内容，通过 title 配置网站的标题、actions 配置快捷链接、features 配置详情介绍，这样我们就拥有了下面的首页样式：

EN



网站快速成型工具

一套为开发者、设计师和产品经理准备的基于
Vue 3 的桌面端组件库

[快速上手](#)[项目简介](#)

简洁至上

以 Markdown 为中心的项目结构，以最少的配置帮助你专注于写作。

Vue 驱动


享受 Vue 的开发体验，可以在 Markdown 中使用 Vue 组件，又可以使用 Vue 来开发自定义主题。

高性能

VuePress 会为每个页面预渲染生成静态的 HTML，同时，每个页面被加载的时候，将作为 SPA 运行。

额外的信息

然后我们进入 docs/.vuepress/ 目录下，新建文件 config.js，这是这个网站的配置页面。下面的代码我们配置了 logo 和导航 navbar，页面顶部右侧就会有首页和安装两个导航。


 复制代码

```

1  module.exports = {
2    themeConfig:{
3      title:"Element3",
4      description:"vuepress搭建的Element3文档",
5      logo:"/element3.svg",
6      navbar:[
7        {
8          link:"/",
9          text:"首页"
10       },{
11         link:"/install",
12         text:"安装"
13       },
14     ]
15   }
16 }
17 }

```

然后我们创建 docs/install.md 文件，点击顶部导航之后，就会显示 install.md 的信息。我们在文稿中就可以直接写上介绍 Element3 如何安装的文档了，下面的文稿就是 Element3 的安装使用说明。

 复制代码

```

1  ## 安装
2  ### npm 安装
3  推荐使用 npm 的方式安装，它能更好地和 [webpack](https://webpack.js.org/) 打包工具配合
4  ```shell
5  npm i element3 -S
6  ```
7  ### CDN
8  目前可以通过 [unpkg.com/element3](https://unpkg.com/element3) 获取到最新版本的资源，
9  ```html
10 <!-- 引入样式 -->
11 <link
12   rel="stylesheet"
13   href="https://unpkg.com/element3/lib/theme-chalk/index.css"
14 />
15 <!-- 引入组件库 -->
16 <script src="https://unpkg.com/element3"></script>
17 ```
18 :::tip
19 我们建议使用 CDN 引入 Element3 的用户在链接地址上锁定版本，以免将来 Element3 升级时受到非
20 :::
21 ### Hello world
22
23 通过 CDN 的方式我们可以很容易地使用 Element3 写出一个 Hello world 页面。[在线演示](http

```

```
24 <iframe height="265" style="width: 100%;" scrolling="no" title="Element3 Demo"
25   See the Pen <a href='https://codepen.io/imjustaman/pen/abZajYg'>Element3 Dem
26   (<a href='https://codepen.io/imjustaman'>@imjustaman</a>) on <a href='https:
27 </iframe>
28 如果是通过 npm 安装，并希望配合 webpack 使用，请阅读下一节：[快速上手](#/zh-CN/compone
29
30
```

然后我们在浏览器里点击安装后，就会看到下图的页面显示。Markdown 已经成功渲染为在线文档了，并且代码也自带了高亮显示。



Element3

首页 安装

安装

npm 安装

CDN

Hello world

安装

npm 安装

推荐使用 npm 的方式安装，它能更好地和 [webpack](#) 打包工具配合使用。

```
1 npm i element3 -S
```

CDN

目前可以通过 unpkg.com/element3 获取到最新版本的资源，在页面上引入 js 和 css 文件即可开始使用。

```
1 <!-- 引入样式 -->
2 <link
3   rel="stylesheet"
4   href="https://unpkg.com/element3/lib/theme-chalk/index.css"
5 />
6 <!-- 引入组件库 -->
7 <script src="https://unpkg.com/element3"></script>
```

TIP

我们建议使用 CDN 引入 Element3 的用户在链接地址上锁定版本，以免将来 Element3 升级时受到非兼容性更新的影响。锁定版本的方法请查看 [unpkg.com](#)。

Hello world

通过 CDN 的方式我们可以很容易地使用 Element3 写出一个 Hello world 页面。[在线演示](#)

HTML

Result

EDIT ON CODEPEN

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <!-- import CSS -->
  <link rel="stylesheet"
href="https://unpkg.com/element3/lib/theme-
chalk/index.css">
</head>
```


Button

然后我们需要在这个文档系统中支持 Element3，首先执行下面的代码安装 Element3：

```
1 npm i element3 -D
```


[复制代码](#)

然后在项目根目录下的 docs/.vuepress 文件夹中新建文件 clientAppEnhance.js，这是 VuePress 的客户端扩展文件。我们导入了 defineClientAppEnhance 来返回客户端的扩展配置。这个函数中会传递 Vue 的实例 App 以及路由配置 Router，我们使用 app.use 来全局注册 Element3 组件，就可以直接在 Markdown 中使用 Element3 的组件了。

 复制代码


```
1
2 import { defineClientAppEnhance } from '@vuepress/client'
3
4 import element3 from 'element3'
5
6 export default defineClientAppEnhance(({ app, router, siteData }) => {
7   app.use(element3)
8 })
```

这样 VuePress 就内置了 Element3。我们在 docs 下面新建 button.md 文件，可以直接在 Markdown 中使用 Element3 的组件进行演示。下面的文稿中我们直接使用了 el-button 组件演示效果。

 复制代码

```
1 ## Button 按钮
2
3 常用的操作按钮。
4 ### 基础用法
5 基础的按钮用法。
6
7 <el-button type="primary">
8 按钮
9 </el-button>
10
11 ```html
12 <el-button type="primary">
13 按钮
14 </el-button>
15 ```
```

然后进入 docs/.vuepress/config.js 中，新增侧边栏 sidebar 的配置之后，就可以看到下图的效果了。

 复制代码

```
1 sidebar:[
```

```
2      {
3        text: '安装',
4        link: '/install'
5      },
6      {
7        text: '按钮',
8        link: '/button'
9      },
10     ]
```



首页 安装

安装

按钮

Button 按钮

常用的操作按钮。

基础用法

基础的按钮用法。

按钮

```
1 <el-button type="primary">
2   按钮
3 </el-button>
```

html

这样我们就基于 VuePress 支持了 Element3 组件库的文档功能，剩下的就是给每个组件写好文档即可。

但是这样的话，el-button 的源码就写了两次，如果我们想更好地定制组件库文档，就需要自己解析 Markdown 文件，在内部支持 Vue 组件的效果显示和源码展示，也就相当于定制了一个自己的 VuePress。

复制代码

```
1 :::demo 使用`type`、`plain`、`round`和`circle`属性来定义 Button 的样式。
2
3 ```html
4 <template>
5   <el-row>
6     <el-button>默认按钮</el-button>
7     <el-button type="primary">主要按钮</el-button>
8     <el-button type="success">成功按钮</el-button>
9     <el-button type="info">信息按钮</el-button>
10    <el-button type="warning">警告按钮</el-button>
11    <el-button type="danger">危险按钮</el-button>
12  </el-row>
```

```
13 </template>
14 ```
15
16 :::
```

它能直接使用下面的`:::demo` 语法，在标记内部代码的同时，显示渲染效果和源码，也就是下图 Element3 官网的渲染效果。

基础用法

基础的按钮用法。



那么接下来我们就看看如何定制，具体操作一下。

解析 Markdown


我们需要自己实现一个 Markdown-loader，对 Markdown 语法进行扩展。

Element3 中使用 Markdown-it 进行 Markdown 语法的解析和扩展。Markdown-it 导出一个函数，这个函数可以把 Markdown 语法解析为 HTML 标签。这里我们需要做的就是

解析出 Markdown 中的 demo 语法，渲染其中的 Vue 组件，并且同时能把源码也显示在组件下方，这样就完成了扩展任务。

Element3 中对 Markdown 的扩展源码都可以在 [🔗 GitHub](#) 上看到。

下面的代码就是全部解析的逻辑：首先我们使用 md.render 把 Markdown 渲染成为 HTML，并且获取内部 demo 子组件；在获取了 demo 组件内部的代码之后，调用 genInlineComponentText，把组件通过 Vue 的 compiler 解析成待执行的代码，这一步就是模拟了 Vue 组件解析的过程；然后使用 script 标签包裹编译之后的 Vue 组件；最后再把组件的源码放在后面，demo 组件的解析就完成了。

 复制代码

```
1  const { stripScript, stripTemplate, genInlineComponentText } = require('./util')
2  const md = require('./config')
3
4  module.exports = function (source) {
5    const content = md.render(source)
6
7    const startTag = '<!--element-demo:'
8    const startTagLen = startTag.length
9    const endTag = ':element-demo-->'
10   const endTagLen = endTag.length
11
12   let componenetsString = ''
13   let id = 0 // demo 的 id
14   const output = [] // 输出的内容
15   let start = 0 // 字符串开始位置
16
17   let commentStart = content.indexOf(startTag)
18   let commentEnd = content.indexOf(endTag, commentStart + startTagLen)
19   while (commentStart !== -1 && commentEnd !== -1) {
20     output.push(content.slice(start, commentStart))
21
22     const commentContent = content.slice(commentStart + startTagLen, commentEnd)
23     const html = stripTemplate(commentContent)
24     const script = stripScript(commentContent)
25
26     const demoComponentContent = genInlineComponentText(html, script)
27
28     const demoComponentName = `element-demo${id}`
29     output.push(`<template #source><${demoComponentName} /></template>`)
30     componenetsString += `${JSON.stringify(
31       demoComponentName
32     )}: ${demoComponentContent},`
33
34     // 重新计算下一次的位置
```

```

35     id++
36     start = commentEnd + endTagLen
37     commentStart = content.indexOf(startTag, start)
38     commentEnd = content.indexOf(endTag, commentStart + startTagLen)
39   }
40
41   // 仅允许在 demo 不存在时，才可以在 Markdown 中写 script 标签
42   // todo: 优化这段逻辑
43   let pageScript = ''
44   if (componenetsString) {
45     pageScript = `
```

```
7     },
8     render(tokens, idx) {
9         const m = tokens[idx].info.trim().match(/^demo\s*(.*)$/);
10        if (tokens[idx].nesting === 1) {
11            const description = m && m.length > 1 ? m[1] : '';
12            const content =
13                tokens[idx + 1].type === 'fence' ? tokens[idx + 1].content : ''
14            return `${md.render(description)}
```

然后我们就实现了 demo-block 组件。接下来我们新建 DemoBlock.vue，在下面的代码中我们通过 slot 实现了组件的渲染结果和源码高亮的效果，至此我们就成功了实现了 Markdown 中源码演示的效果。

[复制代码](#)

```
1 <!-- DemoBlock.vue -->
2 <template>
3   <div class="demo-block">
4     <div class="source">
5       <slot name="source"></slot>
6     </div>
7     <div class="meta" ref="meta">
8       <div class="description" v-if="$slots.default">
9         <slot></slot>
10      </div>
11      <div class="highlight">
12        <slot name="highlight"></slot>
13      </div>
14    </div>
15    <div
16      class="demo-block-control"
17      ref="control"
18      @click="isExpanded = !isExpanded"
19    >
20      <span>{{ controlText }}</span>
21    </div>
22  </div>
```

```
23 </template>
24
25 <script>
26 import { ref, computed, watchEffect, onMounted } from 'vue'
27 export default {
28   setup() {
29     const meta = ref(null)
30     const isExpanded = ref(false)
31     const controlText = computed(() =>
32       isExpanded.value ? '隐藏代码' : '显示代码'
33     )
34     const codeAreaHeight = computed(() =>
35       [...meta.value.children].reduce((t, i) => i.offsetHeight + t, 56)
36     )
37     onMounted(() => {
38       watchEffect(() => {
39         meta.value.style.height = isExpanded.value
40           ? `${codeAreaHeight.value}px`
41           : '0'
42       })
43     })
44
45     return {
46       meta,
47       isExpanded,
48       controlText
49     }
50   }
51 }
52 </script>
```

总结

我们来总结一下今天学到的内容。

首先我们使用 Vue 官网文档的构建工具 VuePress 来搭建组件库文档，VuePress 提供了很好的上手体验，Markdown 中可以直接注册使用 Vue 组件，我们在.vuepress 中可以扩展对 Element3 的支持。

如果我们定制需求更多一些，就需要自己解析 Markdown 并且实现对 Vue 组件的支持了，我们可以使用 Markdown-it 插件解析，支持 Vue 组件和代码高亮，这也是现在 Element3 文档的渲染方式。

思考题

最后留给你一道思考题：现在很多组件库开始尝试使用 Storybook 来搭建组件库的文档，那么这个 Storybook 相比于我们实现的文档有什么特色呢？

欢迎你在评论区分享你的看法，也欢迎你把这节课的内容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 3  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 25 | 表格：如何设计一个表格组件

下一篇 27 | 自定义渲染器：如何实现Vue的跨端渲染？

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

写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。