**【全栈FootPrints-02】组件化与重构网站布局页面**

[](https://xiaozhuanlan.com/u/giscafer)

[giscafer](https://xiaozhuanlan.com/u/giscafer)已关注

预计阅读时间27分钟14 天前

本篇为 **旅游足迹网站（FootPrints）** 第 2 篇

上篇介绍了布局页面的开发，但是只是一个布局的模板代码简单的实现，这在真正的项目开发过程中，不推荐这么随意的写代码的，因为随着时间的推移，页面代码的复杂度和维护成本越来越高，所以在开发的时候，就要合理的抽离独立的子模块组件化，以编写可维护、可扩展的代码为目标。

本篇将介绍如何组件化开发，以及通过组件化的方式重构布局页面的代码。

**组件化**

**概念介绍**

组件化的意思就是将某个功能封装组件，在前端，组件可以分为基础功能组件、通用UI组件、基础业务组件等。我们平时用的社区组件库就是通用UI组件，比如 Vue 的 Element UI、Ant Design Vue 等；前端的三大库（框架）都有 ant design 版本的 UI 组件库。

而基础功能组件与业务组件则更多是在项目开发的过程，对一些常用的页面UI进行封装成组件，以便实现复用提高效率，不需要重复的开发同样的功能。比如电商类应用常见的地址选择组件。

由多个组件（相同或者不同）组成的某个局部UI就成了 **区块** ，然后由区块组成的页面，如果是很常用的页面就成了模板。而区块和模板标准化之后，并聚合起来就形成了物料市场。物料其实就是包含组件、区块、页面模板的统称。

**组件封装标准规范**

为什么 Ant Design 会有 React、Vue、Angular 版本的组件库，原因是因为 Antd 拥有好看的样式和交互设计，是由蚂蚁的体验团队精心设计的成果。学习前端组件化的时候，需要认识一些标准规范，清楚规范后，你在封装组件的时候尽量做到自己当时能做到最好的。

组件封装标准规范有哪些？

**交互&视觉规范**

和上边提到的 Ant Design 设计规范一样，封装第一个组件，第二个组件总要统一的一个交互标准规范，不然两个组件放在一起，会不伦不类。Ant Design 这类组件库的开源样式比较标准，如果是基于 antd 系列的就依照该规范去封装组件即可。本系列文章都是基于 Element。

**一致性考量**

一致性就是保证组件的交互和样式效果都保存统一的风格，避免特殊化，这块其实也是在交互&视觉规范定下了。开发的时候按照此标准即可。

**粒度控制**

组件封装时拆分的粒度大小，合理的拆分组件粒度，使得组件的每个元素可以灵活的使用。这块可以去学习参考开源UI组件库的源码，看一些组件是如何拆分成细小粒度再封装的。比如 select 下拉选择组件，option 会被单独封装成子组件，你会发现option子组件在其他组件封装的时候也使用到了。保证了重复的单元抽离封装，重复利用和解耦。

**向前兼容性**

指的是以前版本的软件（组件）支持现在版本的数据（方法、属性），现在版本支持以后版本的数据（方法、属性）。组件封装的时候，如果修改了逻辑，可能就是 **Break Change（破坏性升级）** ，导致的是老版本组件想使用新版本组件的时候，升级了会引起其他问题，不能使用等情况。

**向后扩展性**

同理 **向前兼容性**，向后扩展性也叫向后兼容性，指的是现在的版本可以支持以前的版本数据。

**可复用性**

组件封装原本就是为了复用的目的，不然就不是一个合格的组件封装，或者不应该成为基础功能组件、UI组件或者业务组件了。在组件封装的时候，做到高内聚低耦合，避免像复用的时候牵扯到别的东西而导致不可用的情况。比如一个地址选择组件，目的就是封装好，选择省市县区街道的数据，可以自定义选择到哪个级别。有个业务场景是，选择市区数据的时候，触发一个商品查询，如果这个逻辑写在了地址组件中，别的使用场景就可能无法利用这个组件了。

**可测试性**

组件一般都建议写组件测试的，因为组件是重复利用的UI单元，很多页面都用到，并且可能用到组件的不同的特性，保证组件功能没有被后续的迭代维护改坏了，影响到别的场景的情况，一个完整的测试是必备的。而组件测试的前提是，这个组件封装的时候，是具备可测试性的。

同样的组件 如果你的测试反复运行，同样的数据却不能产生相同的结果，要么是代码有问题，要么是测试有问题。

**可定制性**

根据组件的需求，尽量能满足可定制性，比如select 支持自定义下拉模板渲染、tree 组件支持自定义node节点模板渲染等

**文档沉淀&可读性&及时性**

组件封装好后，应该同步输出一个组件使用文档说明，并且列举出组件的属性和方法、事件等特性。就如同开源的 UI 组件库一样，有个详细的组件文档，避免开发人员在使用这个组件的时候还需要去阅读组件源码才知道有哪些可以用的属性、方法和事件等，这样效率极低。组件文档也需要保证及时的同步更新，并有很好的可读性，程序员的文档都是 Markdown 写的，Markdown 可以轻松的实现可读性较好的文档布局。

**其他更多**

组件封装其实也有很多规范和约束，这里就不一一的列举说明了，也可以结合一些设计模式，比如开闭原则等。想学习更多就去搜索相关的资源了解。

**重构布局页面代码**

组件化已经大概的介绍完了，现在进行实战练习。以下是代码讲解，前提是需要你掌握 Vue 和了解 Nuxt.js，如果你不知道到此是做什么，请阅读前一篇文章 [【FootPrints-01】Nuxt.js 搭建网站基础开发框架](https://xiaozhuanlan.com/topic/8031764592)

上篇我们把布局代码都堆在了 /layouts/default.vue 文件下了，现在我们进行规范重构一下，基础功能组件化抽离。在着手组件化之前，要先思考怎么设计，如果你有经验，可以直接在大脑思考直接就编写代码了，但如果你没有经验，最后用笔和纸写一下，怎么差分组件，代码结构如何设计等。

**子组件拆分**

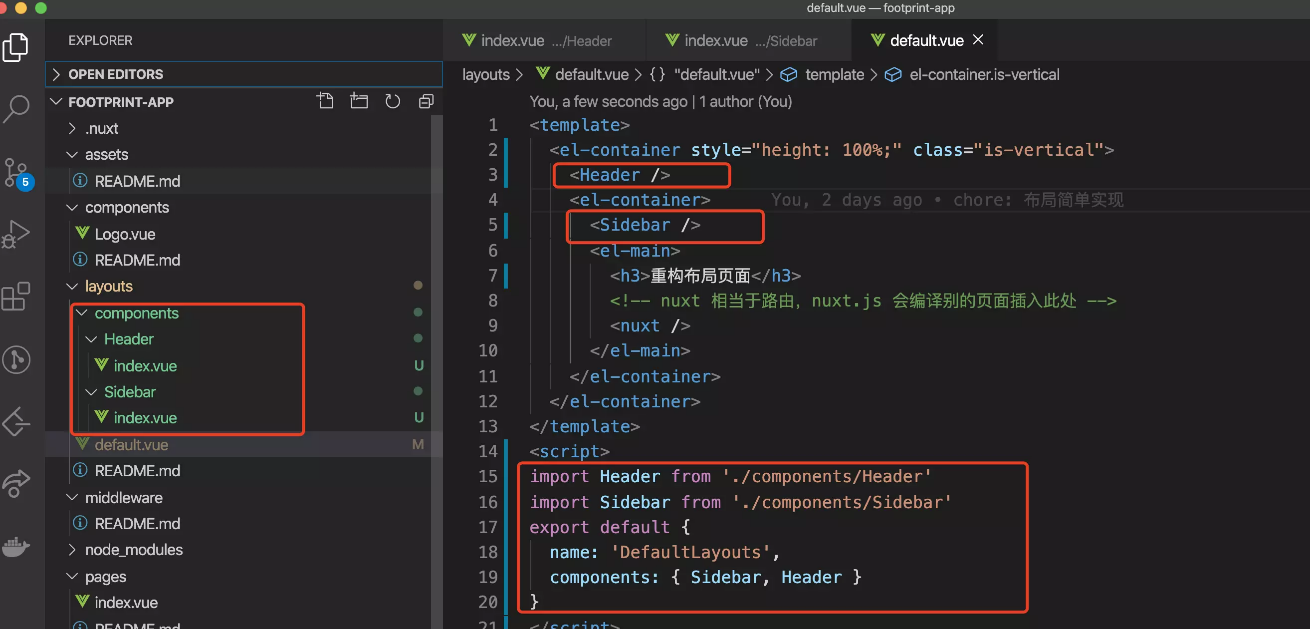
layouts 的文件夹目录结构是这样的，default.vue 文件里放了网站的布局模板代码

1. ├── layouts *// layouts 目录中的每个文件 (顶级) 都将创建一个可通过页面组件中的 layout 属性访问的自定义布局。*
2. │ ├── README.md
3. │ └── **default**.vue

重构设计后是这样的：

1. ```js
2. ├── layouts
3. │ ├── README.md
4. │ └── Header *// header头部*
5. │ │ └── index.vue
6. │ │
7. │ └── Sidebar *// 左侧菜单 aside*
8. │ │ └── index.vue
9. │ │
10. │ └── **default**.vue *// 引入Header 组件和 Sidebar 组件*

设计好后，我们将目录和代码文件创建好，将Header 和 Sidebar 代码独立出来成子组件，如图



组件拆分

这样效果还是我们上次的页面的效果。

**Header 组件增强和完善**

Header 目前的代码如下：

1. <template>
2. <el-header style="text-align: right; font-size: 12px">
3. <el-dropdown>
4. <i class="el-icon-setting" style="margin-right: 15px"></i>
5. <el-dropdown-menu slot="dropdown">
6. <el-dropdown-item>查看</el-dropdown-item>
7. <el-dropdown-item>新增</el-dropdown-item>
8. <el-dropdown-item>删除</el-dropdown-item>
9. </el-dropdown-menu>
10. </el-dropdown>
11. <span>giscafer</span>
12. </el-header>
13. </template>
14. <script>
15. **export** **default** {
16. name: 'Header',
17. props: {},
18. data() {
19. **return** {}
20. },
21. created() {},
22. mounted() {},
23. methods: {}
24. }
25. </script>
26. <style lang="scss" scoped>
27. .el-header {
28. background: #1b2034;
29. color: #fff;
30. line-height: 60px;
31. }
32. </style>

这只是单纯的代码模板，接下来我们进行改造成网站可用的样子。



Logo子组件和头像等

改造后的代码为：

1. <template>
2. <el-header style="text-align: right; font-size: 12px">
3. <Logo /> *<!-- Logo 子组件 -->*
4. <div class="right-menu">
5. <el-dropdown class="avatar-container right-menu-item hover-effect" trigger="click">
6. <div class="avatar-wrapper">
7. <img src="/images/avatar.png" class="user-avatar" />
8. <i class="el-icon-caret-bottom"></i>
9. </div>
10. <el-dropdown-menu slot="dropdown">
11. <router-link to="/profile/index">
12. <el-dropdown-item>账号信息</el-dropdown-item>
13. </router-link>
14. <el-dropdown-item divided @click.native="logout">
15. <span style="display:block;">退出登陆</span>
16. </el-dropdown-item>
17. </el-dropdown-menu>
18. </el-dropdown>
19. </div>
20. </el-header>
21. </template>
22. <script>
23. **import** Logo **from** './Logo'
24. **export** **default** {
25. name: 'Header',
26. components: { Logo },
27. props: {},
28. data() {
29. **return** {}
30. },
31. created() {},
32. mounted() {},
33. methods: {
34. logout() {
35. **this**.$message('你点击了退出')
36. }
37. }
38. }
39. </script>
40. <style lang="scss" scoped>
41. .el-header {
42. // 样式多省略了……
43. }
44. </style>

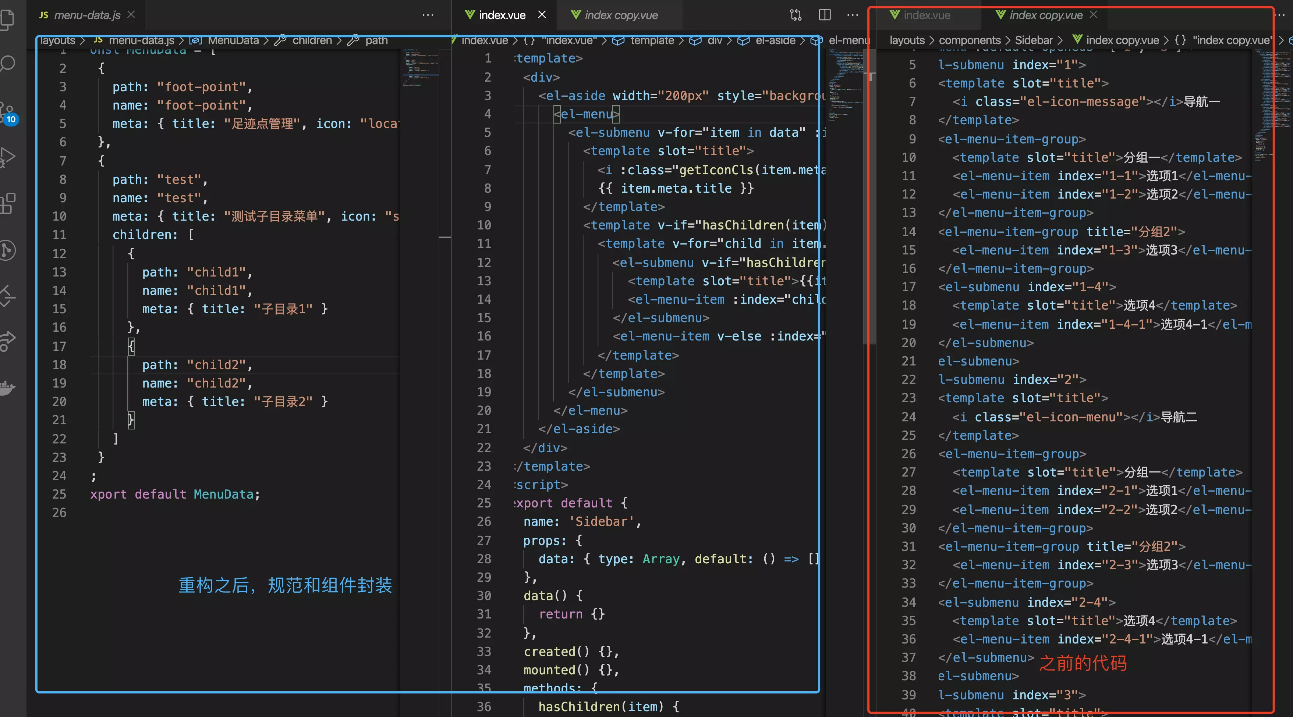
**Sidebar 重构组件完善**

左侧的菜单导航要结合路由使用的场景重构，支持配置化，自动渲染是最好的效果。所以将 Sidebar 组件封装成通过菜单数据 menuData 来渲染出菜单效果。

1. <template>
2. <div>
3. <el-aside width="200px" style="background-color: rgb(238, 241, 246)">
4. <el-menu>
5. <el-submenu v-for="item in data" :index="item.name" :key="item.name">
6. <template slot="title">
7. <i :class="getIconCls(item.meta.icon)"></i>
8. {{ item.meta.title }}
9. </template>
10. <template v-if="hasChildren(item)">
11. <template v-for="child in item.children">
12. <el-submenu v-if="hasChildren(child)" :index="child.name" :key="child.name">
13. <template slot="title">{{item.meta.title}}</template>
14. <el-menu-item :index="child.name" :key="child.name">{{child.meta.title}}</el-menu-item>
15. </el-submenu>
16. <el-menu-item v-else :index="child.name" :key="child.name">{{ child.meta.title }}</el-menu-item>
17. </template>
18. </template>
19. </el-submenu>
20. </el-menu>
21. </el-aside>
22. </div>
23. </template>
24. <script>
25. **export** **default** {
26. name: 'Sidebar',
27. props: {
28. data: { type: Array, default: () => [] }
29. },
30. data() {
31. **return** {}
32. },
33. created() {},
34. mounted() {},
35. methods: {
36. hasChildren(item) {
37. **return** item.children && item.children.length > 0
38. },
39. getIconCls(iconName) {
40. **return** `el-icon-${iconName}`
41. }
42. }
43. }
44. </script>
45. <style lang="scss" scoped>
46. .el-aside {
47. color: #333;
48. }
49. </style>

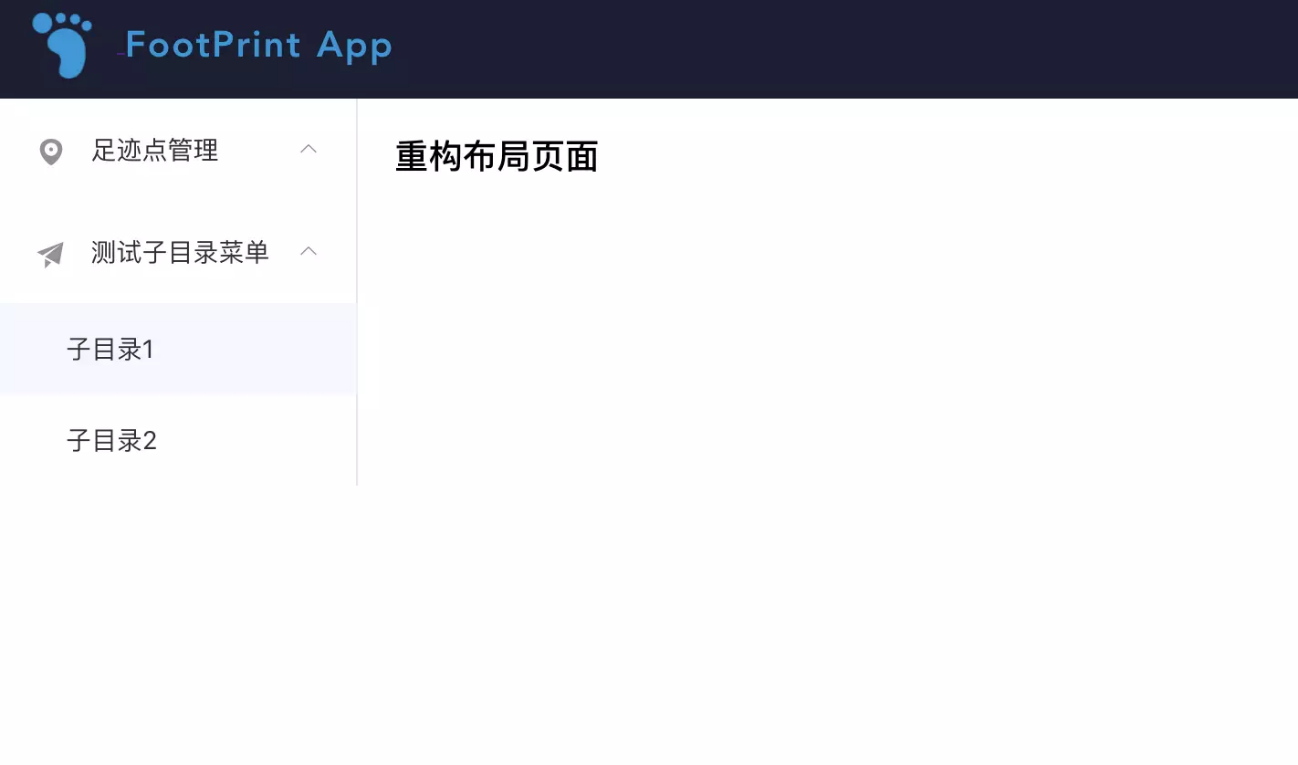
data 数据为：layouts/menu-data.js ，之后在开发页面的时候，新增菜单，只需要在此文件配置即可。

1. **const** MenuData = [
2. {
3. path: "foot-point",
4. name: "foot-point",
5. meta: { title: "足迹点管理", icon: "location" }
6. },
7. {
8. path: "test",
9. name: "test",
10. meta: { title: "测试子目录菜单", icon: "s-promotion" },
11. children: [
12. {
13. path: "child1",
14. name: "child1",
15. meta: { title: "子目录1" }
16. },
17. {
18. path: "child2",
19. name: "child2",
20. meta: { title: "子目录2" }
21. }
22. ]
23. }
24. ];
25. **export** **default** MenuData;



重构前和重构后对比

**最终效果**：



重构后根据数据渲染的左侧菜单

**注意事项** ：项目中我们用了 SCSS 作为样式预处理语言，所以需要安装相关依赖，执行：npm i node-sass sass-loader --save-dev

**小结**

好了，以上就是本篇的主要内容了。本篇学习主要是学习什么是组件化，组件封装需要有哪些规范和约定，以及实战中重构上一章节网站布局页面代码，抽离子组件、封装独立，提供使用的便利性和规范代码等。如果有哪些你希望深入了解的地方还未涉及到，希望你可以留言给我或者在微信群中讨论。

本文代码改动git记录见[commit/400ac75](https://github.com/gis520/footprint-app/commit/400ac75bb9c4db1375dba1bb59405babb1b06416)，之前的文章：