

**【全栈FootPrints-04】路由跳转与前端列表页如何高效开发**

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

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

预计阅读时间26分钟5 天前

本篇为 **旅游足迹网站（FootPrints）** 第 4 篇

上篇介绍了 Nuxt.js 中如何使用 Leaflet 开发地图，本篇将讲解如何配置路由跳转与前端列表页如何高效开发。

**Nuxt.js 的路由**

给小白的话：路由是一个网站导航到不同页面的方式，任何框架开发的网站，都需要路由来跳转页面，不管是静态的链接地址，还是动态的转发地址，都属于路由跳转。

Nuxt.js 的路由系统学习了解可以看官方的文档-[路由](https://zh.nuxtjs.org/guide/routing/)

Nuxt.js 的路由是基于 vue-router 实现的， Nuxt.js 通过 pages 目录的结构，动态地生成路由的配置。也就是不需要额外配置路由文件，直接新建文件夹和文件，Nuxt.js 会在编译时自动生成路由配置。在 html 代码中，使用 <nuxt-link> 标签可以实现路由导航，接下来实操介绍一下。

接着上篇的代码中，我们在 pages 目录下新建目录和文件 footprint/index.vue ，如图



footprint页面

接下来，配置路由，使得可以跳转到 footprint 页面，找到第二篇 [【全栈FootPrints-02】组件化与重构网站布局页面](https://xiaozhuanlan.com/topic/7396251804) 下我们重构和组件化的左侧菜单组件抽离的菜单数据配置文件 layouts/menu-data.js，修改足迹点管理菜单的路由配置属性 path 为 footprint。

1. {
2. path: "footpoint", *// 这个地方*
3. name: "foot-point",
4. meta: { title: "足迹点管理", icon: "location" }
5. }

第二步，需要调整 Sidebar 组件的模板，加入 <nuxt-link> 标签，修改 layouts/component/Sidebar/index.vue，以下仅为 **html 部分代码** ：

1. <template>
2. <div style="100%">
3. <el-aside width="200px">
4. <el-menu>
5. <template v-for="item in data">
6. *<!-- 有子菜单，嵌套菜单 -->*
7. <el-submenu v-if="hasChildren(item)" :index="item.name" :key="item.name">
8. <template slot="title">
9. <i :class="getIconCls(item.meta.icon)"></i>
10. {{ item.meta.title }}
11. </template>
12. <template v-for="child in item.children">
13. <el-submenu v-if="hasChildren(child)" :index="child.name" :key="child.name">
14. <template slot="title">{{item.meta.title}}</template>
15. <el-menu-item :index="child.name" :key="child.name">
16. *<!-- 这里 -->*
17. <nuxt-link :to="child.path">{{ child.meta.title }}</nuxt-link>
18. </el-menu-item>
19. </el-submenu>
20. <el-menu-item v-else :index="child.name" :key="child.name">
21. *<!-- 这里 -->*
22. <nuxt-link :to="child.path">{{ child.meta.title }}</nuxt-link>
23. </el-menu-item>
24. </template>
25. </el-submenu>
26. <template v-else>
27. *<!-- 只有一层菜单 -->*
28. <el-menu-item :index="item.name" :key="item.name">
29. <i :class="getIconCls(item.meta.icon)"></i>
30. *<!-- 这里 -->*
31. <nuxt-link :to="item.path">{{ item.meta.title }}</nuxt-link>
32. </el-menu-item>
33. </template>
34. </template>
35. </el-menu>
36. </el-aside>
37. </div>
38. </template>

最后我们点击**足迹点管理**菜单即可跳转到刚才新建的足迹点页面了。



到此，简单的路由就实现了，也顺便完善了 Sidebar 菜单组件，日后前端页面开发只需要做两步：

* 1、在 pages/ 目录下新建页面；
* 2、修改 layouts/menu-data.js 菜单配置。（注意在pages新增页面文件需要重新启动才生效，因为要编译路由 npm run dev）

**前端列表页如何高效开发**

**现状分析**

后台管理系统，大部分的页面可能都是列表查询页面和表单页面。列表页面有以下一些特点：

* 有表单输入框作为查询条件的输入
* 有个表格展示数据
* 点击查询按钮，进行接口请求
* 获取到查询结果更新表格数据展示
* 表格有分页，点击分页可以根据页数进行翻页查询数据。

不管列表页面如何设计样式或者交互，基本都离不开上边的这几个功能特性。简单的认为，一个列表页面开发需要写这 5 个功能代码，多个列表页面开发就需要重复的写 n\*5 遍，本质上一样的代码，只是字段名称，接口名称不一样。所以，开发多个类似的列表页面就是重复的 **苦力活！！！**

对于苦力活，用编程来解决有两种方式： **一是自动化（智能化）生成代码；二是组件化、封装工具方法，实现高度复用提效**

**进一步封装表格组件，实现高效开发**

UI组件库 Element UI 或 Ant Design 都提供了表格组件，基础的UI组件目的是灵活，如果你封装得过多，灵活性就可能丢失了。项目开发时，有哪些组件可能用上或者页面的交互方式是怎么样的，在项目开始阶段就确定了，所以，很多时候我们在项目开发的过程中，都可以对UI组件库进行二次的封装，根据自己的业务特性进行高度封装，以便提高效率。（更好的方式是，团队或者公司统一规范，业务组件统一封装，多个系统都可以统一使用了，同样的组件，多个项目不需要重复封装多次）

本项目开发使用的是 Element UI，Element UI 的表格组件和分页组件其实就是分开的两个组件，你可以在开发之前，将这两个组件合并封装起来。思路是包装分页组件 Pagination，然后封装 Table 组件，Table 组件引入 Pagination 组件再高度封装出一个方便使用的表格组件，下面是组件代码介绍：

**分页组件**

由于代码量过大，这里省略 Pagination 组件的代码，下载代码本地看或者在线预览 [components/Pagination/index.vue](https://github.com/gis520/footprint-app/blob/master/components/Pagination/index.vue)。

**为什么需要封装分页组件？** 因为每个公司和每个团队的后端架构定的规范可能不一样，使得分页接口的那些分页信息 pageInfo 字段和结构可能不一样，如果不包装一下，你在使用UI组件时，也需要转换一下字段。转换的方式是不推荐的，这样每个人写代码都需要转换一次，多余的代码和多余的开发浪费，所以需要针对你的项目或者团队，封装一次分页组件也差不多是一劳永逸了。

本项目后续进行后端开发时，分页的接口定的返回规范为：

1. *// 分页信息*
2. pageInfo: {
3. totalSize: Number, *// 总记录数*
4. totalPages: Number, *// 总页数*
5. size: Number, *// 每页多少条记录*
6. first: Boolean, *// 是否是第一页*
7. last: Boolean, *// 是否是最后一页*
8. number: Number *// 第几页*
9. }

这里的分页组件也是根据这个数据结构进行封装的，以便提供给表格组件直接关联使用。

**表格组件**

表格组件 Table 组件的完整代码： [components/Table/index.vue](https://github.com/gis520/footprint-app/blob/master/components/Table/index.vue)

Table 组件的 HTML 代码：

1. <template>
2. <div class="custom-table">
3. <el-table
4. :key="tableKey"
5. v-loading="loading"
6. :data="tableData"
7. :cell-class-name="cellClassName"
8. :span-method="spanMethod"
9. :max-height="maxHeight"
10. :empty-text="emptyText"
11. :row-key="rowKey"
12. border
13. fit
14. :highlight-current-row="highlightCurrentRow"
15. style="width: 100%;"
16. @selection-change="handleSelectionChange"
17. >
18. <el-table-column v-if="selection" type="selection" width="55" align="center"></el-table-column>
19. *<!-- columns -->*
20. <template v-if="columns && columns.length>0">
21. <el-table-column
22. v-for="col in columns"
23. :key="col.field"
24. :label="col.label"
25. :width="col.width"
26. :min-width="col.minWidth"
27. :align="col.align || 'center'"
28. :show-overflow-tooltip="col.overflow"
29. >
30. <template slot-scope="{row}">
31. <span v-if="!col.scopedSlots">{{ row[col.field] }}</span>
32. *<!-- custom render -->*
33. <slot v-else :name="col.scopedSlots.customRender" :row="row"></slot>
34. </template>
35. </el-table-column>
36. </template>
37. *<!-- action template -->*
38. <slot></slot>
39. <slot name="action"></slot>
40. </el-table>
41. <pagination
42. v-show="showPagination && pageInfo.totalSize>0"
43. :total="pageInfo.totalSize"
44. :page.sync="innerCurrentPage"
45. :limit.sync="pageSize"
46. :page-sizes="pageSizes"
47. @pagination="queryChange"
48. />
49. </div>
50. </template>

通过传参的方式接收 columns 字段数组描述信息，遍历生成表格，在使用组件的时候，可以减少html的书写，趋向于数据驱动UI的方式，使用起来有点像 AntdV 了。

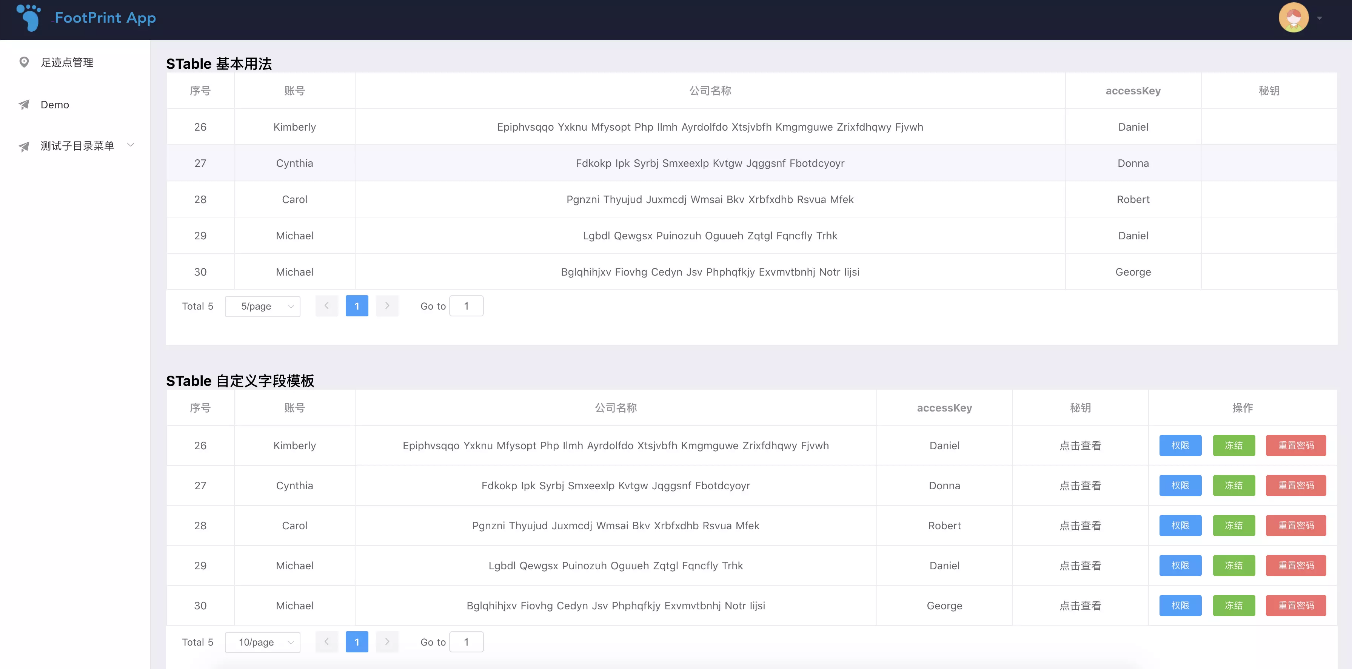
**通过Demo看封装的STable组件如何使用**

新建了 pages/demo/index.vue 文件，里边写了 STable 的使用demo

最简单的表格实现代码只有这么多，并且如果没有别的东西，不需要写一行样式

1. <template>
2. <s-table :data="data" :loading="listLoading" :columns="columns" @query-change="getList">
3. </s-table>
4. </template>
5. <script>
6. **import** STable **from** '@/components/Table'
7. **export** **default** {
8. components: { STable },
9. data() {
10. **return** {
11. columns: [
12. {
13. label: '序号',
14. field: 'id',
15. width: 90
16. },
17. {
18. label: '账号',
19. field: 'account',
20. width: 160
21. },
22. {
23. label: '公司名称',
24. field: 'companyName'
25. },
26. {
27. label: 'accessKey',
28. field: 'accessKey',
29. width: 180
30. },
31. {
32. label: '秘钥',
33. field: 'secret',
34. width: 180
35. }
36. ],
37. data: null,
38. listLoading: false,
39. listQuery: {}
40. }
41. },
42. created() {},
43. methods: {
44. handleFilter() {
45. **this**.listQuery.page = 1
46. **this**.getList()
47. },
48. getList(evt) {
49. **if** (evt) {
50. Object.assign(**this**.listQuery, evt)
51. }
52. **this**.listLoading = true
53. *// 请求后端*
54. fetchList(**this**.listQuery).then(response => {
55. **this**.data = response
56. **this**.listLoading = false
57. })
58. }
59. }
60. }
61. </script>

可以看到， **字段只需要配置** 就遍历生成了表头，然后剩下就是接口请求数据，如果有查询输入框和查询难，就是上面代码中 this.listQuery 绑定即可。代码量降到了最低，可以对比 Element 官方文档的写法看看哪种方式更高效。



简单表格和自义字段渲染效果截图

**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 Nuxt.js 路由以及介绍如何高效开发后台管理系统的列表页面。高效终究都是组件化的结果，通过组件封装提高复用率，减少重复的代码编写。

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。

本文代码改动的 git 记录见[commit/2e8936](https://github.com/gis520/footprint-app/commit/2e89365d1752e36dfb51ab590ec1c54519a3f088)，源码：[footprint-app](https://github.com/gis520/footprint-app)